TÉCNICO+
FORMAÇÃO AVANÇADA

Deep Learning

Gonçalo M. Correia

GALP 2023

Gonçalo M. Correia

goncalo.correia@priberam.pt

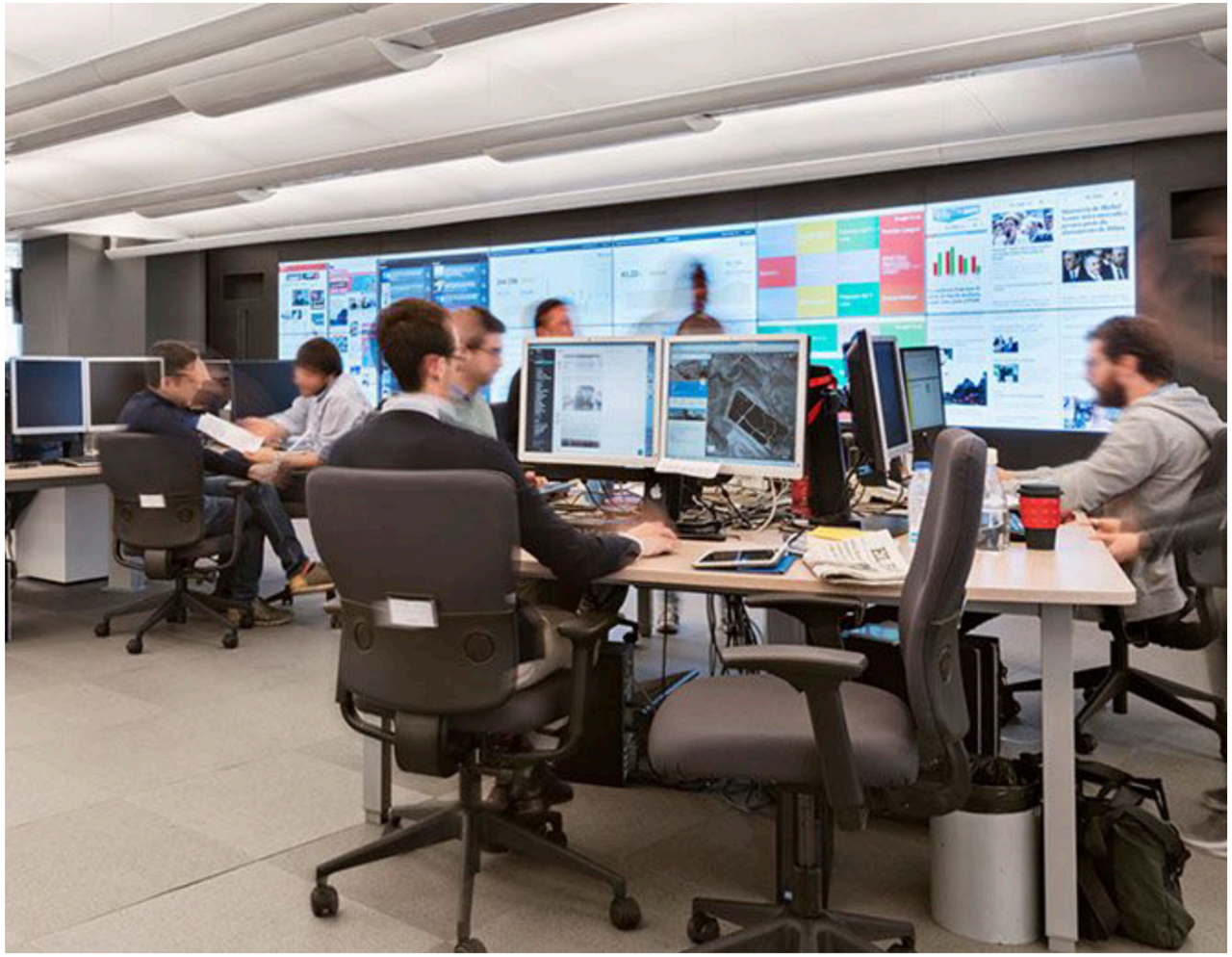
- Senior Research Scientist at Priberam
- Invited Assistant at IST Lisboa

- Work in A.I.:
  - Machine learning;
  - Natural Language Processing;
  - Neural networks.

Priberam Labs

NLP Solutions

- Research team: we conduct applied and exploratory research in the areas of Machine Learning and Natural Language Processing
- Media Intelligence
- Healthcare
- Legal
- Client-specific solutions using NLP



The DL module

Contents

Introduction to deep learning

Learning with neural networks

Convolutional neural networks

Recurrent neural networks

Representation learning

Logistics

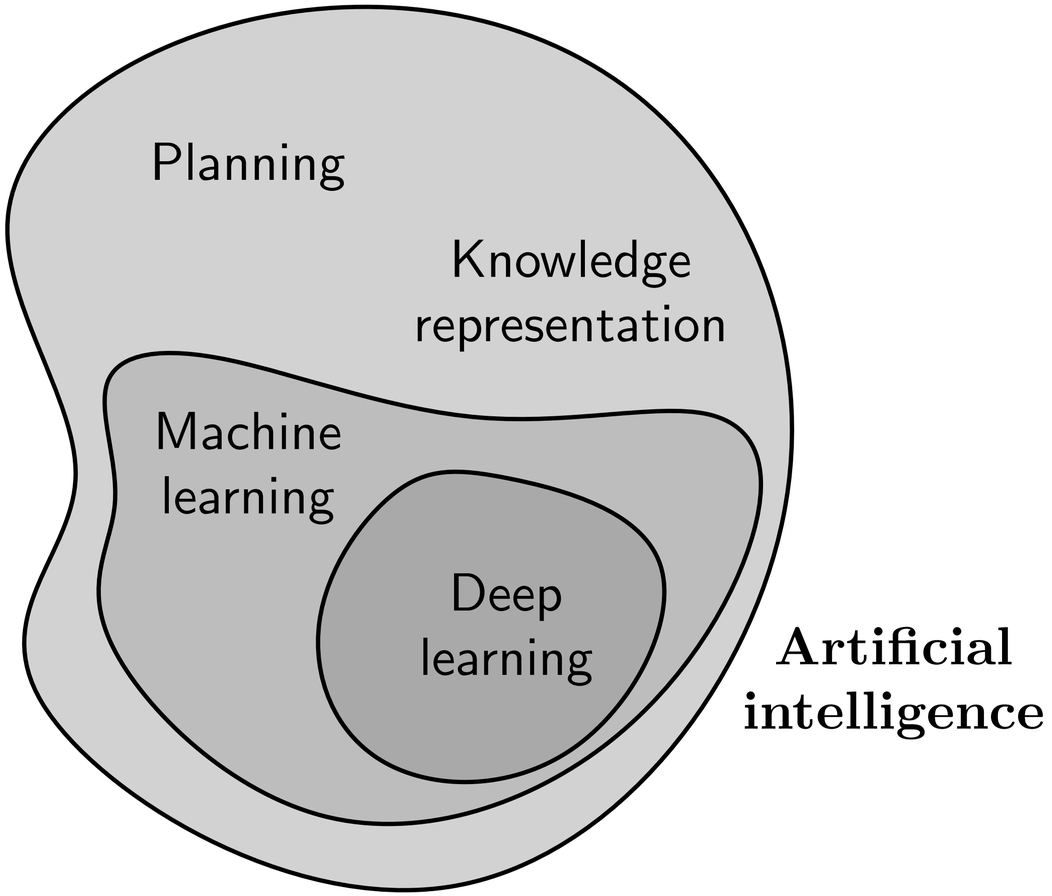
- **Five dates:**
  - May 23
  - May 24
  - May 25
  - May 31
  - June 1
- **Two sessions/day:**
  - 10h00 to 12h30
  - 14h00 to 16h30
- **Morning session:**
  - Two parts, with a short break
  - Mostly theoretical
- **Afternoon session:**
  - Two parts, with a short break
  - Mostly practical (“lab”)

Introduction to DL

DL and artificial intelligence

One and the same?

- Deep learning is part of the broader field of machine learning…
- … which, in turn, is part of the area of artificial intelligence

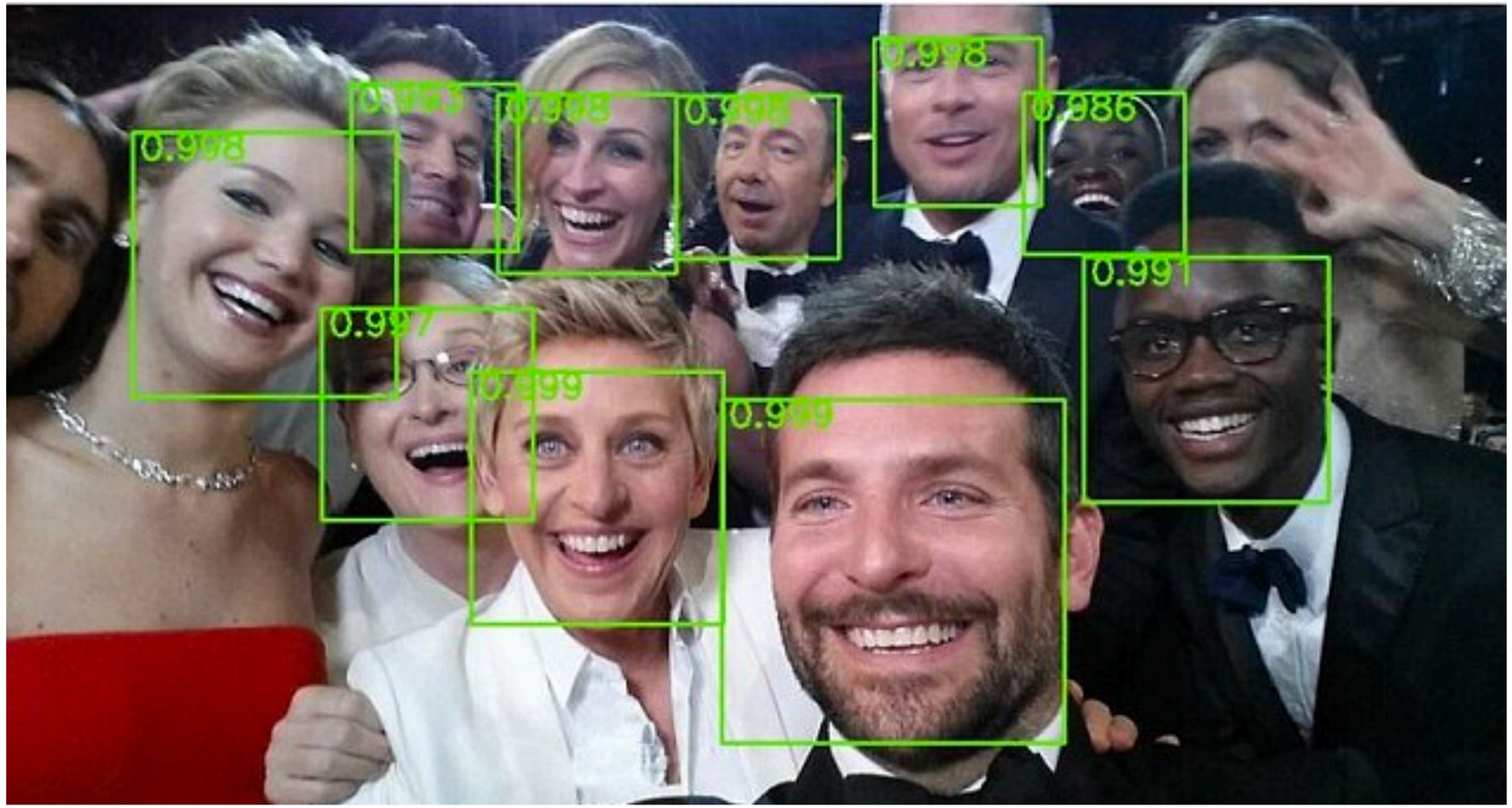


Recent successes in A.I.

fueled by D.L. breakthroughs

Deep learning success stories

Face detection



Farfade et al, ICMR 2015

Deep learning success stories

Image captioning

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Xu et al, ICML 2015

Deep learning success stories

Speech processing

Microsoft's new breakthrough: AI that's as good as humans at listening... on the phone

Microsoft's new speech-recognition record means professional transcribers could be among the first to lose their jobs to artificial intelligence.

Written by **Liam Tung**,
Contributor

on October 19, 2016 | Topic: Innovation

Xiong et al, ICASSP, 2017

Deep learning success stories

Machine translation

Google unleashes deep learning tech on language with Neural Machine Translation

Devin Coldewey @techcrunch / 9:14 PM GMT+1 • September 27, 2016

Comment

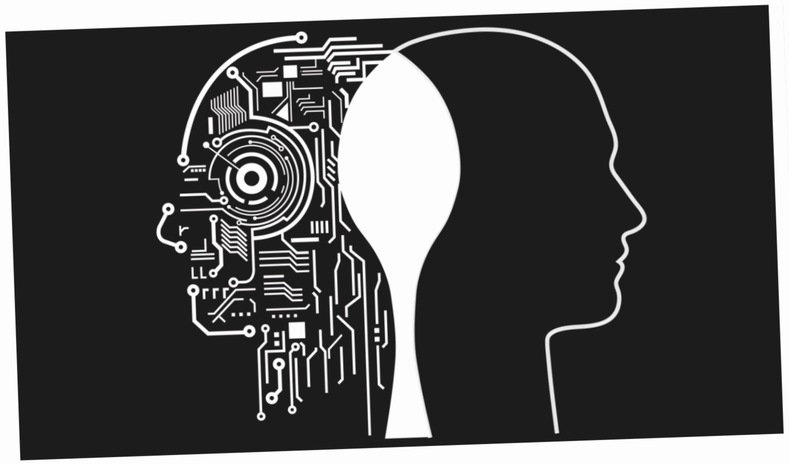


Image Credits: razum / Shutterstock

Translating from one language to another is hard, and creating a system that does it automatically is a major challenge, partly because there are just so many words, phrases and rules to deal with. Fortunately, neural networks eat big, complicated data sets for breakfast.

Wu et al, arXiv, 2016

Deep learning success stories

Text generation

A robot wrote this entire article. Are you scared yet, human?

GPT-3

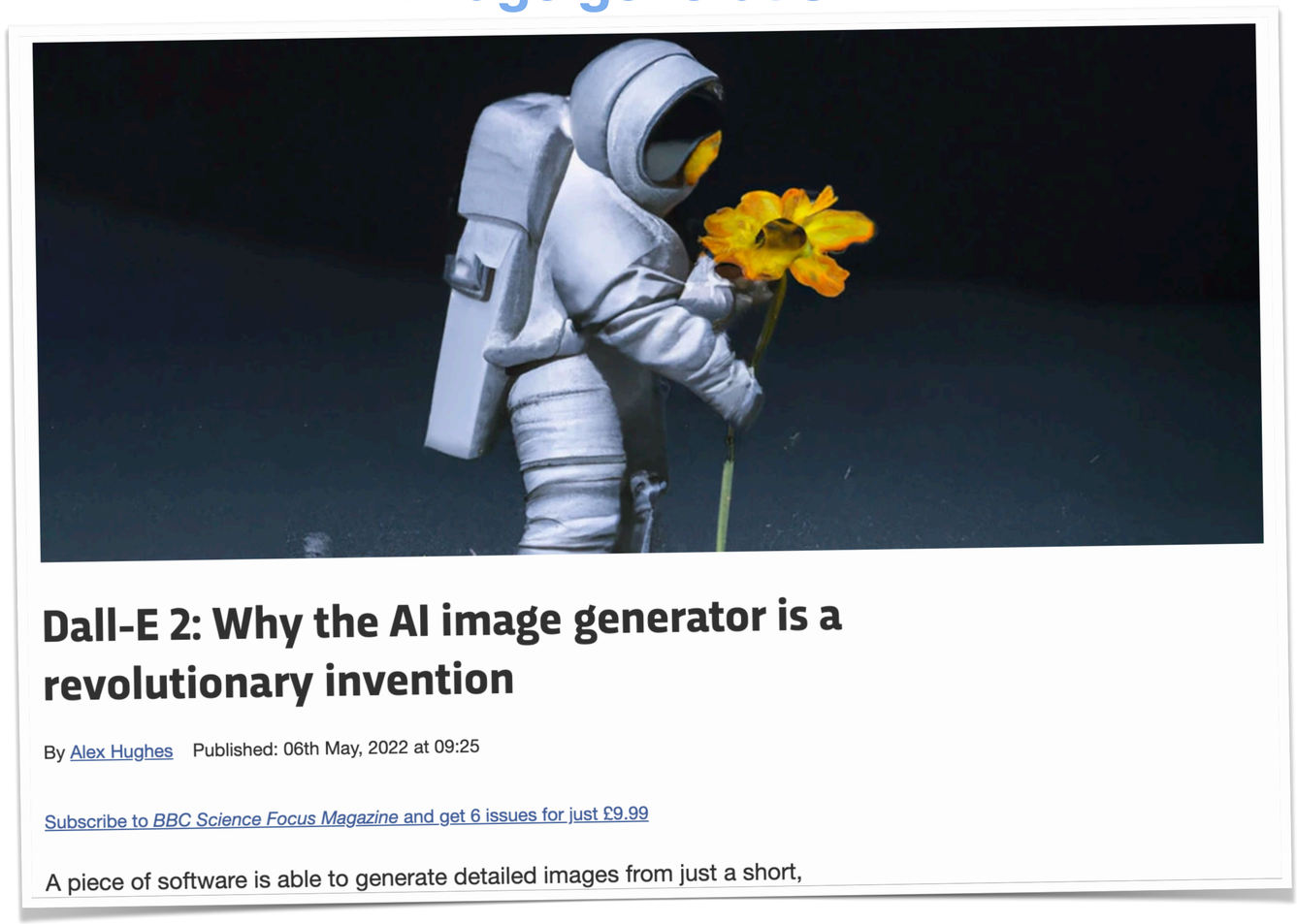
We asked GPT-3, OpenAI's powerful new language generator, to write an essay for us from scratch. The assignment? To convince us robots come in peace

For more about GPT-3 and how this essay was written and edited, please read our editor's note below

Radford et al, OpenAI, 2019

Deep learning success stories

Image generation



Dall-E 2: Why the AI image generator is a revolutionary invention

By Alex Hughes Published: 06th May, 2022 at 09:25

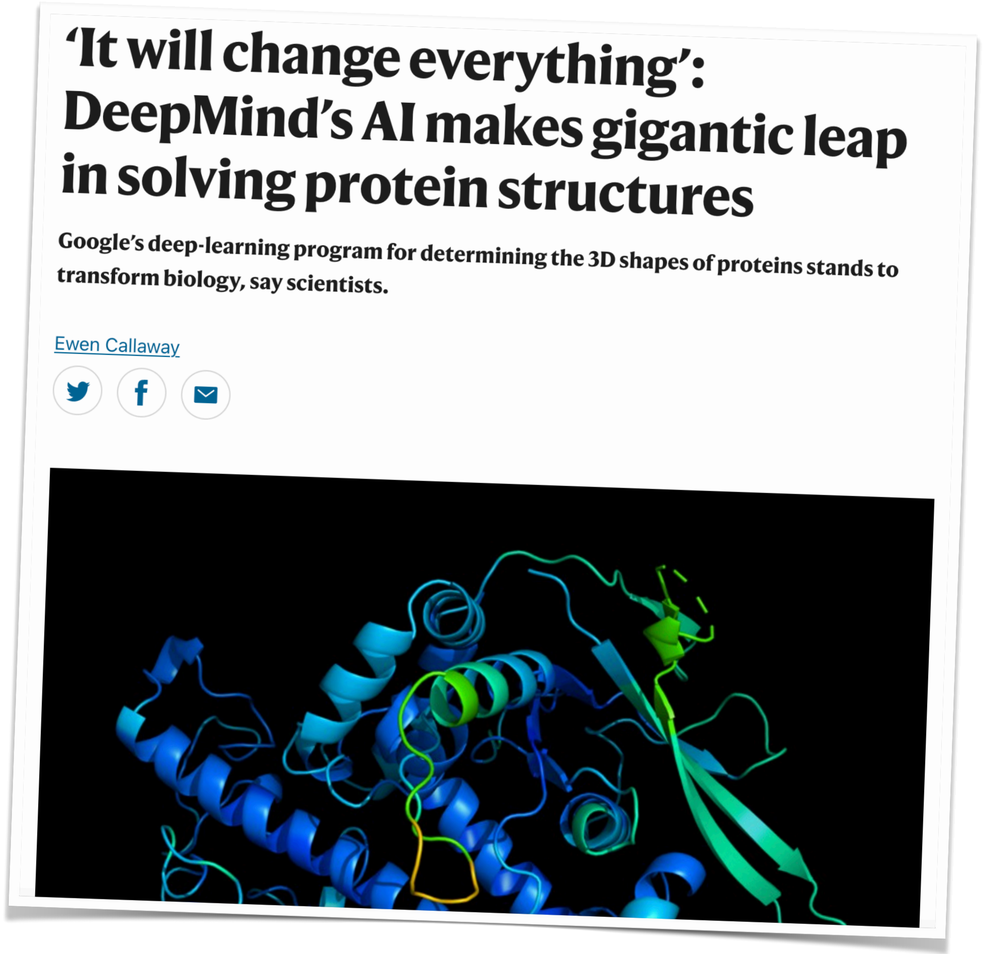
Subscribe to *BBC Science Focus Magazine* and get 6 issues for just £9.99

A piece of software is able to generate detailed images from just a short,

Ramesh et al, arXiv, 2022

Deep learning success stories

Protein folding



‘It will change everything’: DeepMind’s AI makes gigantic leap in solving protein structures

Google’s deep-learning program for determining the 3D shapes of proteins stands to transform biology, say scientists.

Ewen Callaway

Jumper et al, Nature, 2021

Deep learning success stories

Game playing 1

Schrittwieser et al, Nature, 2020

Deep learning success stories

Game playing 2

DeepMind's StarCraft-playing AI beats 99.8 per cent of human gamers

TECHNOLOGY 30 October 2019

By Donna Lu

Vinyals et al, Nature, 2019

But what is deep learning?

What is Deep Learning?

- Neural networks?
- Neural networks with many layers?
- Anything beyond shallow (linear) models for machine learning?
- Anything that learns representations?
- A form of learning that is really intense and profound?

What is Deep Learning?

- Neural networks
- Neural networks with many layers
- Anything beyond shallow (linear) models for machine learning
- Anything that learns representations
- A form of learning that is really intense and profound

Why now?

Neural networks have been around since the 50s…

- Confluence of several factors:
  - Abundance of data
  - Increase of computational power
  - Better software engineering tools (e.g., auto-differentiation engines)
  - Some specialized algorithmic innovations (better activation functions, novel regularization strategies, novel architectures)

Learning with neural networks

Learning ~~with~~ neural networks

Machine learning

"*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.*"

Tom Mitchell

Summarizing: Improve performance in given task using data ("experience").

Machine learning

- To define a machine learning problem, we must identify:
  - The task
  - The performance measure
  - The data

Supervised learning

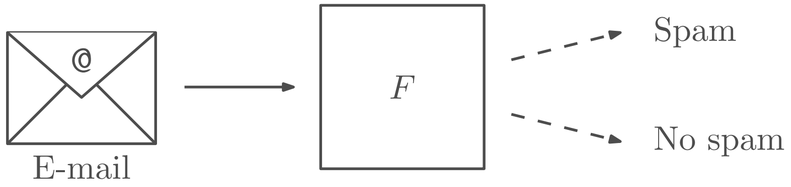
… or learning from examples

- The task:
  - Determine a relation *F* that maps a set $\mathcal{X}$ of possible inputs (e.g., a set of e-mails) to a set $\mathcal{Y}$ of possible outputs (e.g., {spam, no spam})
- The data:
  - Set of input-output pairs (“examples”), $\mathcal{D} = \{(x_n, y_n), n = 1, \ldots, N\}$
- The performance:
  - Measure how well does *F* capture the input-output relation implicit in $\mathcal{D}$

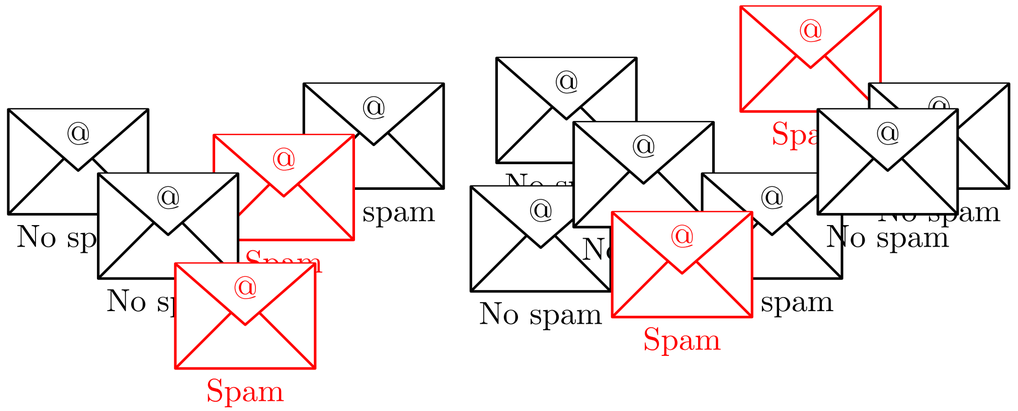
Supervised learning

Training a spam filter

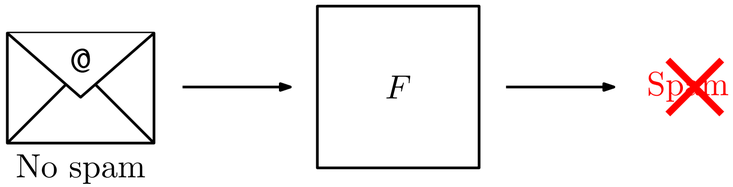
- The task:



- The data:

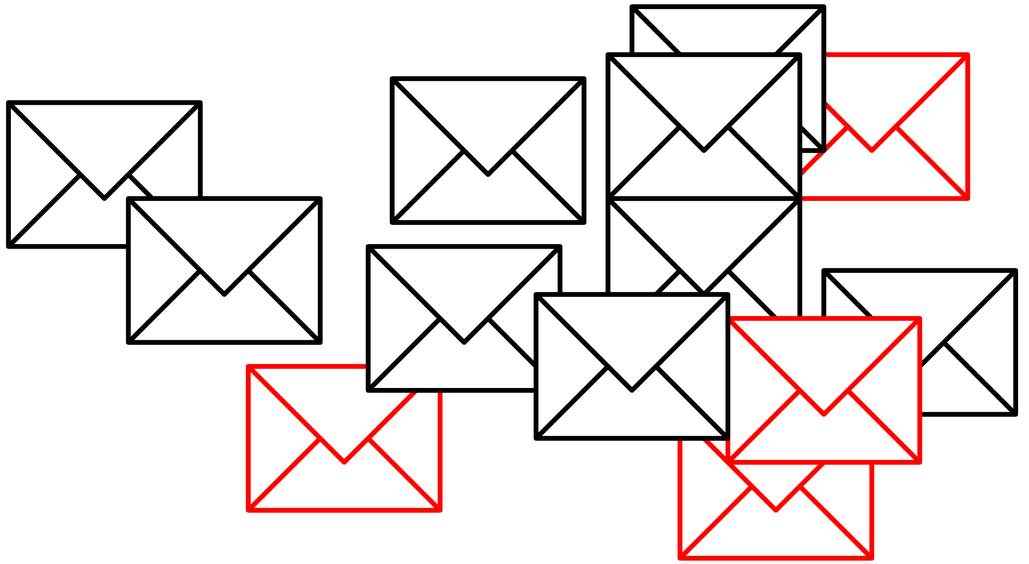


- The performance:



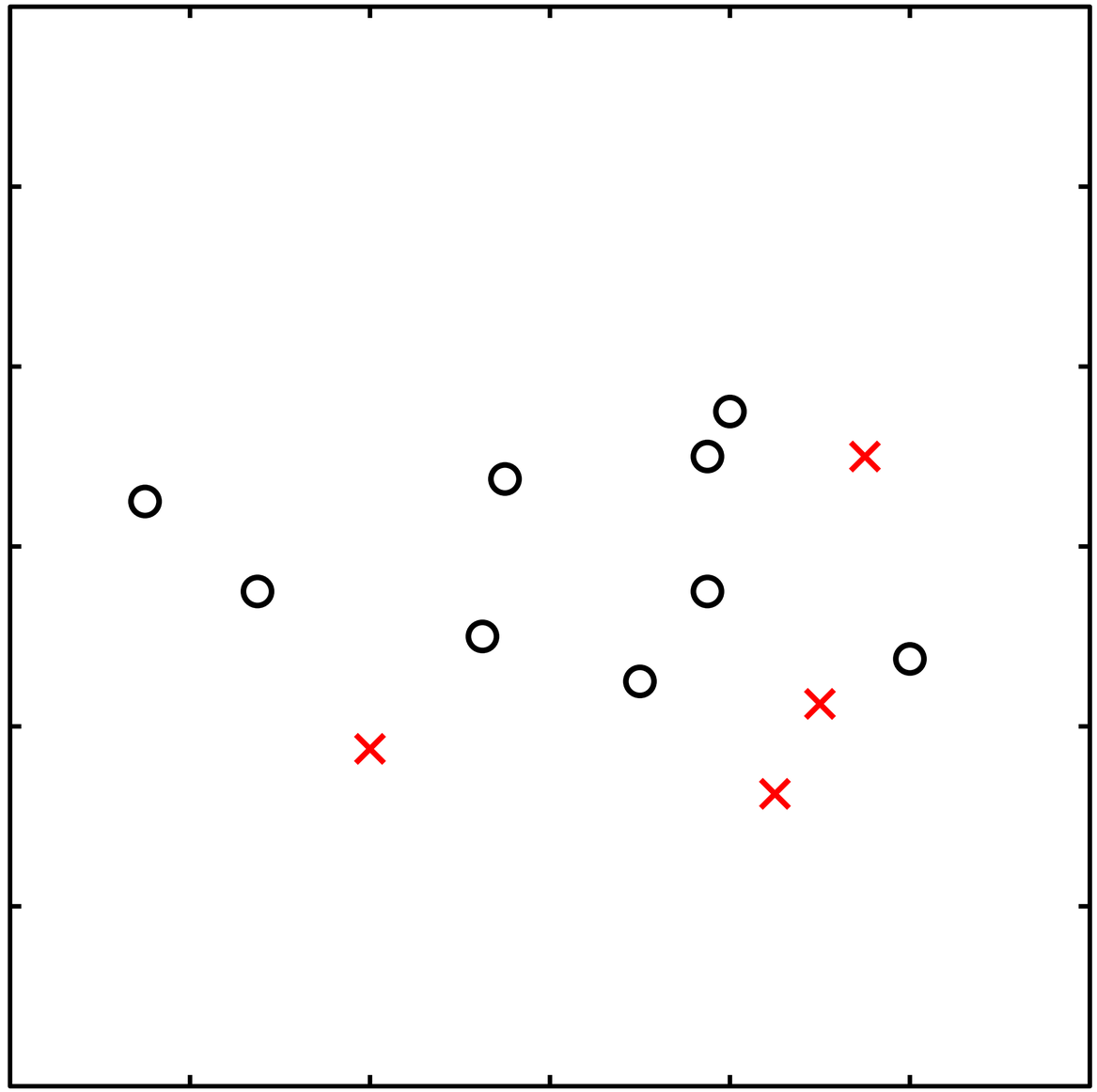
How does it work?

- Let’s say we can describe our inputs (mails) using two numbers



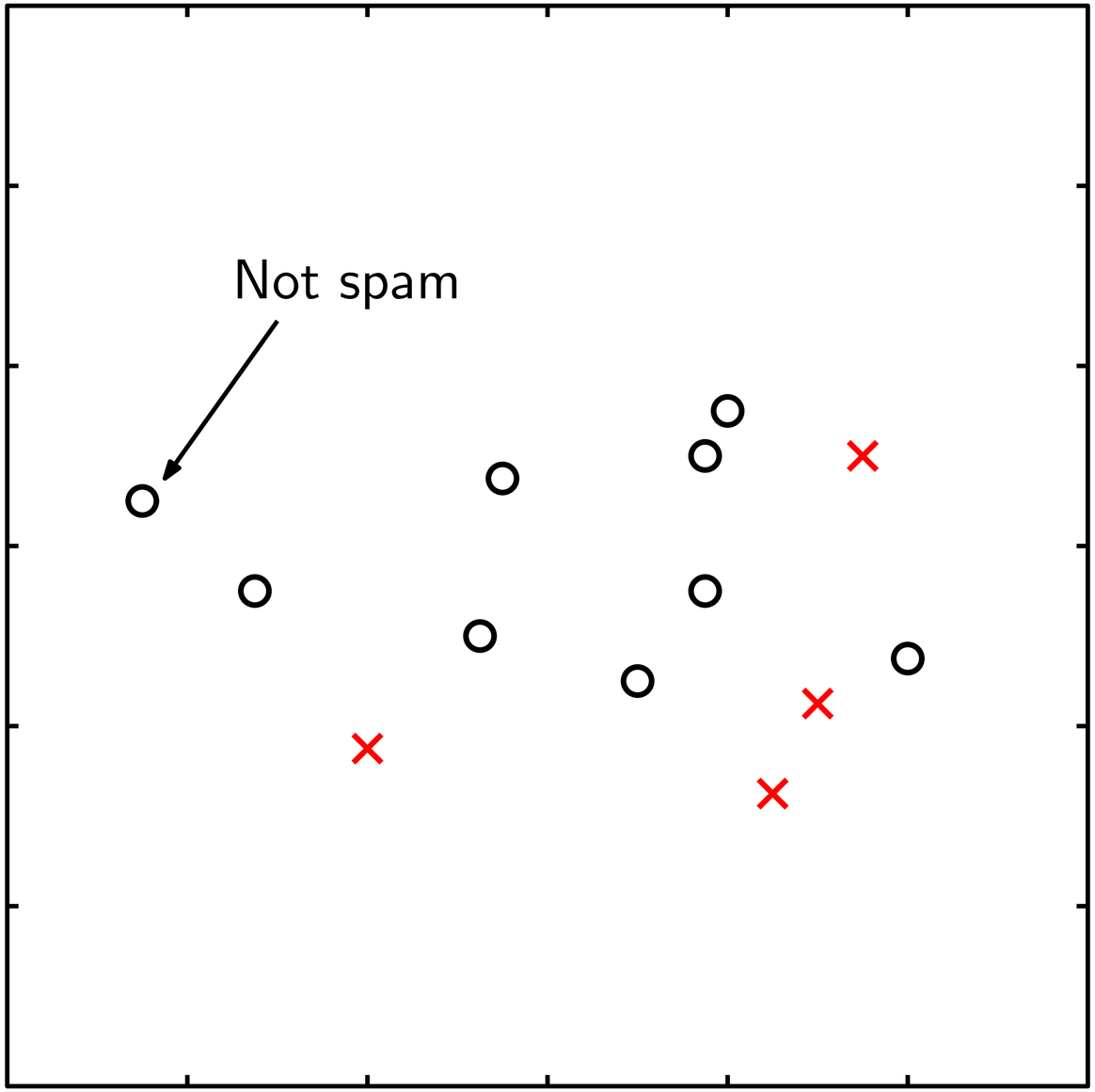
How does it work?

- Let’s say we can describe our inputs (mails) using two numbers



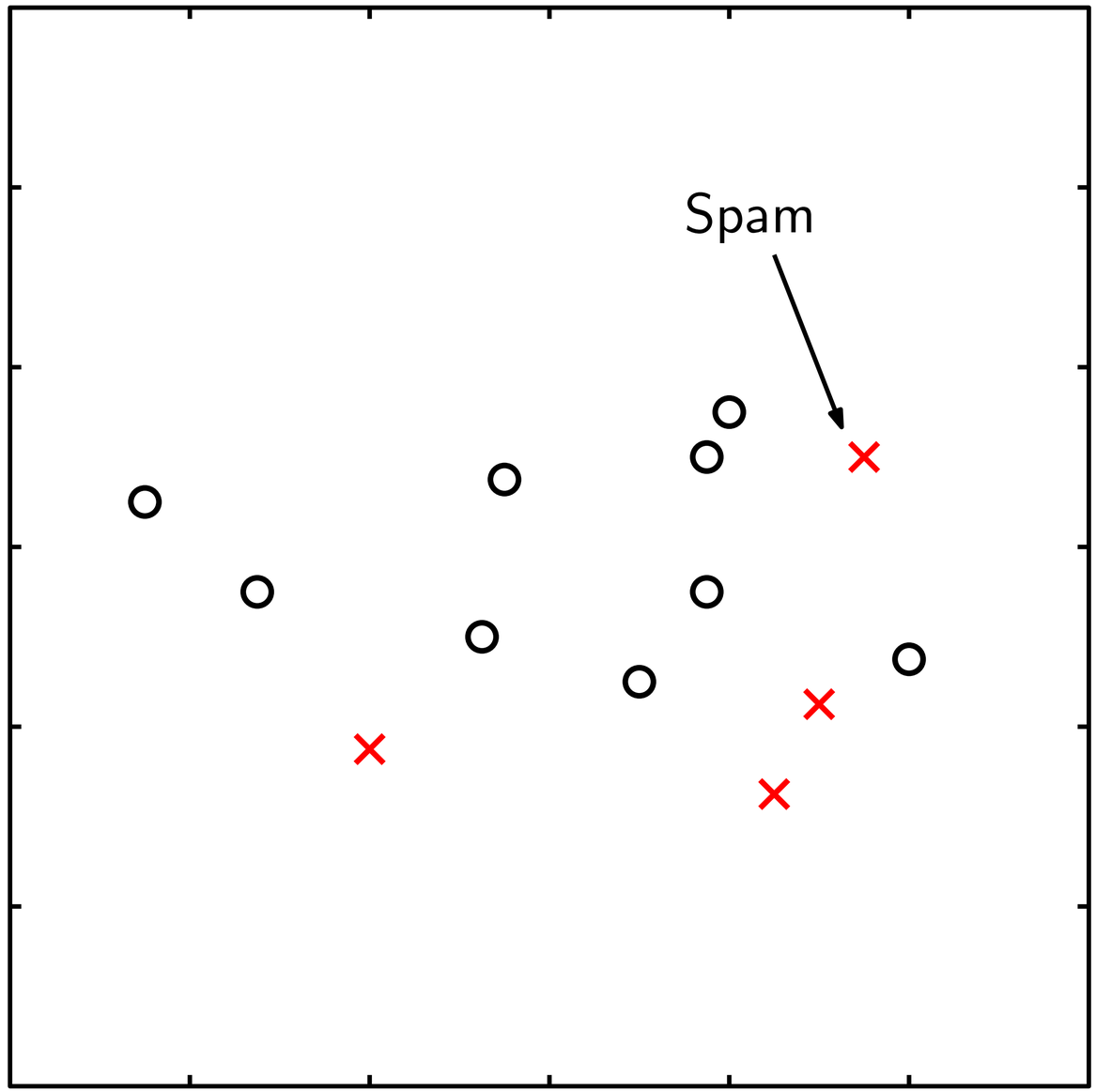
How does it work?

- Let’s say we can describe our inputs (mails) using two numbers



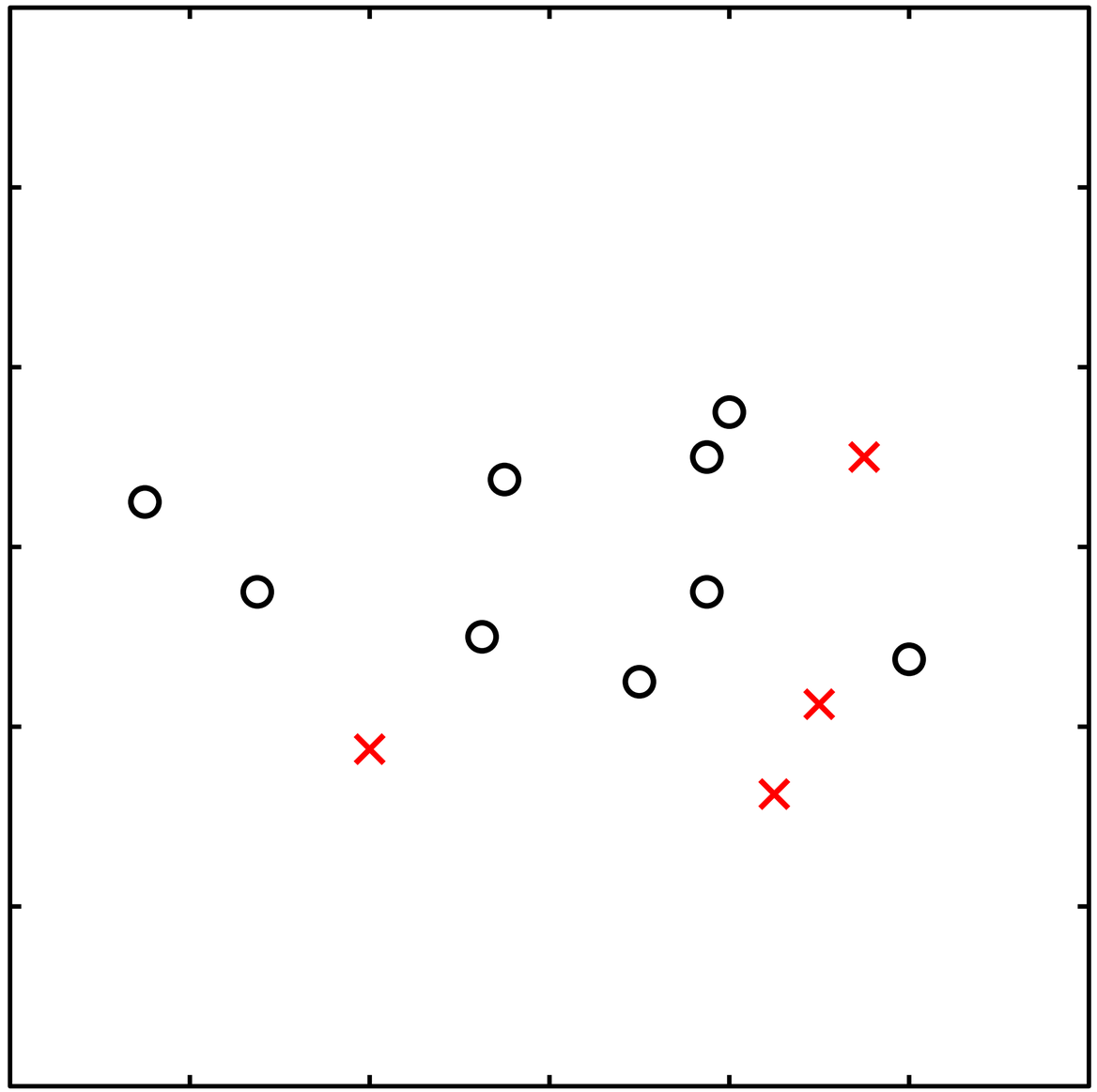
How does it work?

- Let’s say we can describe our inputs (mails) using two numbers



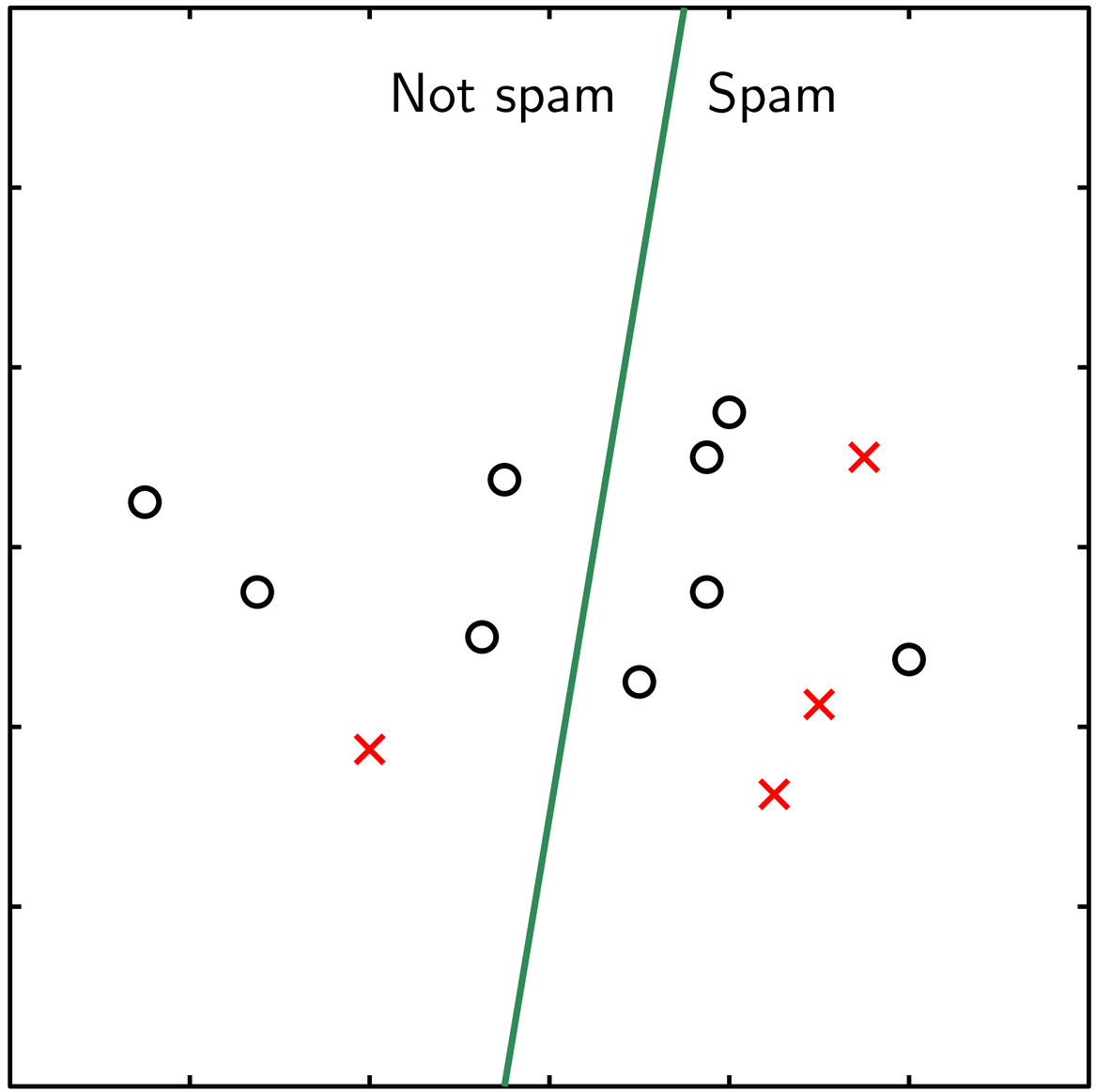
How does it work?

- Let’s say we can describe our inputs (mails) using two numbers
- We want to come up with a way to separate the circles from the crosses



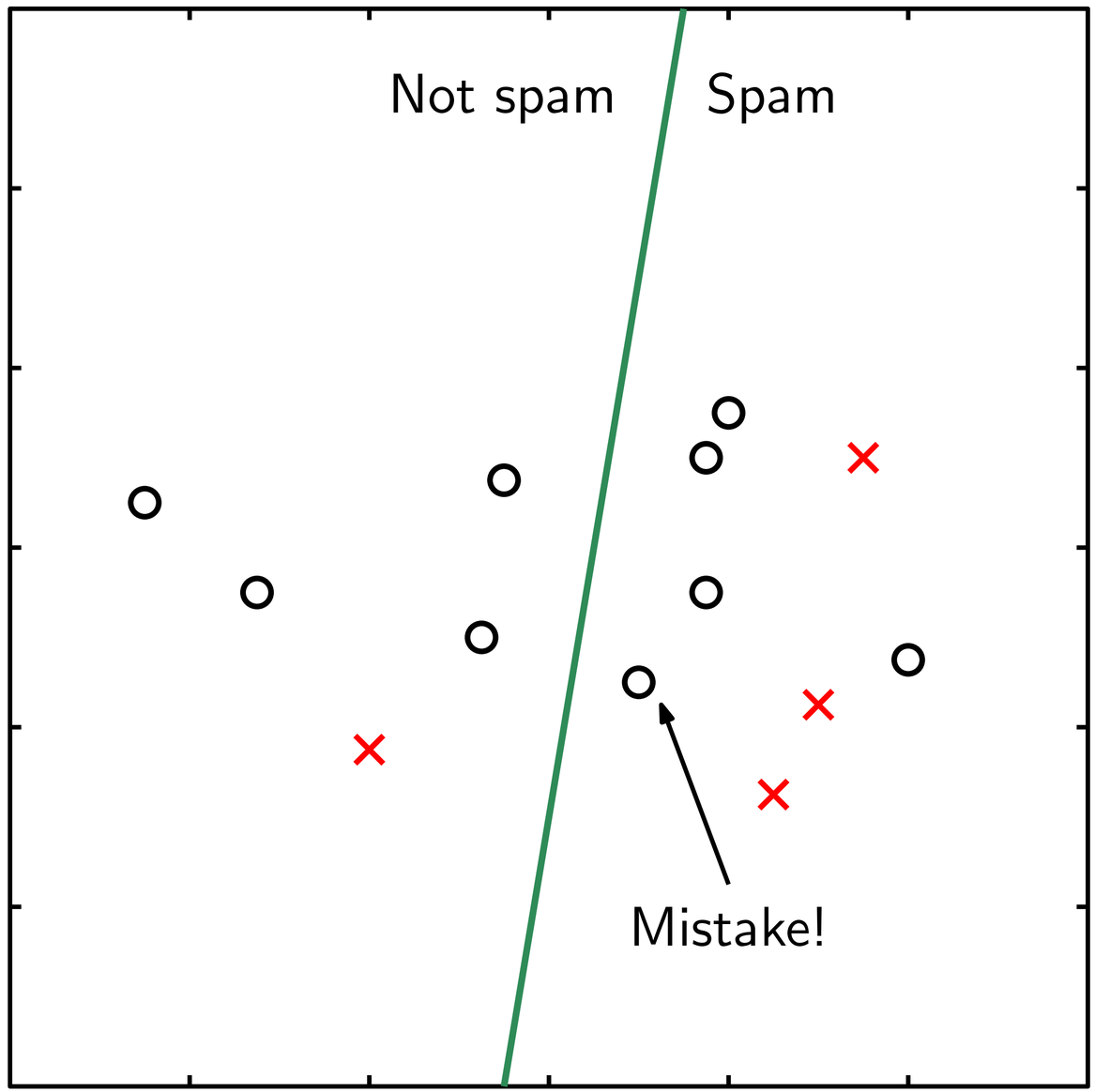
How does it work?

- Let’s say we can describe our inputs (mails) using two numbers
- We want to come up with a way to separate the circles from the crosses
- We slowly adjust our “separation rule” to correct for mistakes



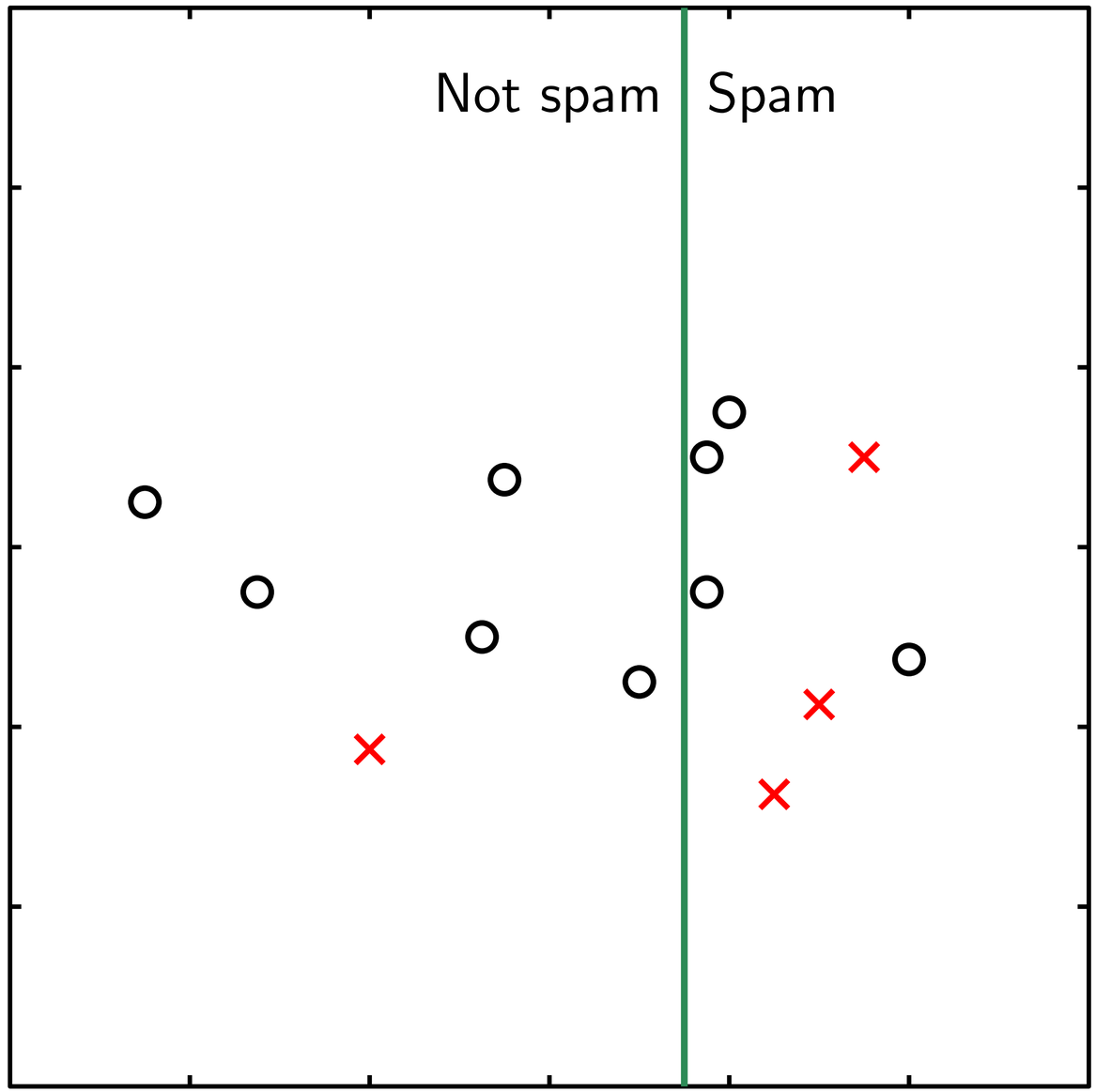
How does it work?

- Let’s say we can describe our inputs (mails) using two numbers
- We want to come up with a way to separate the circles from the crosses
- We slowly adjust our “separation rule” to correct for mistakes



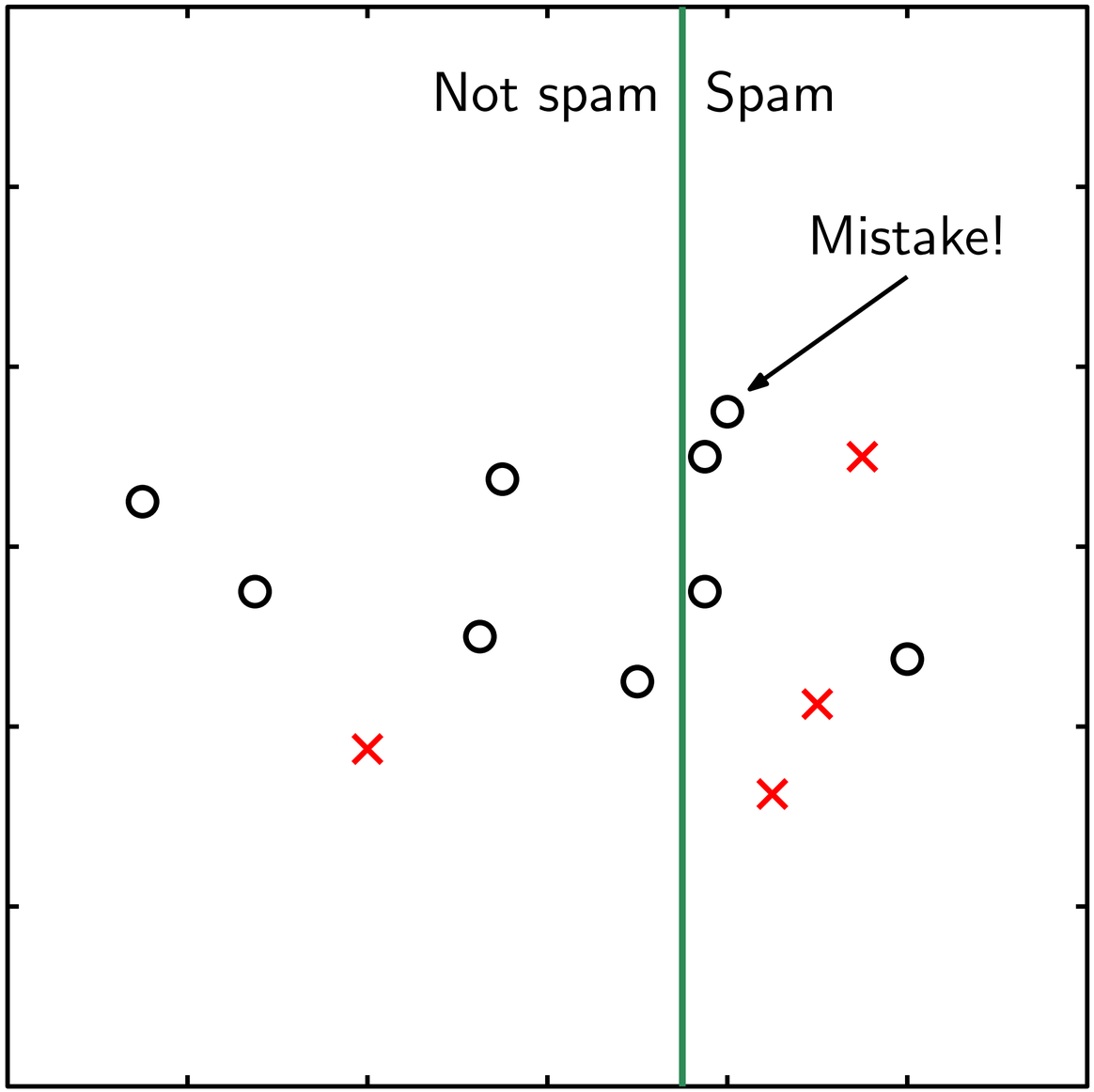
How does it work?

- Let’s say we can describe our inputs (mails) using two numbers
- We want to come up with a way to separate the circles from the crosses
- We slowly adjust our “separation rule” to correct for mistakes



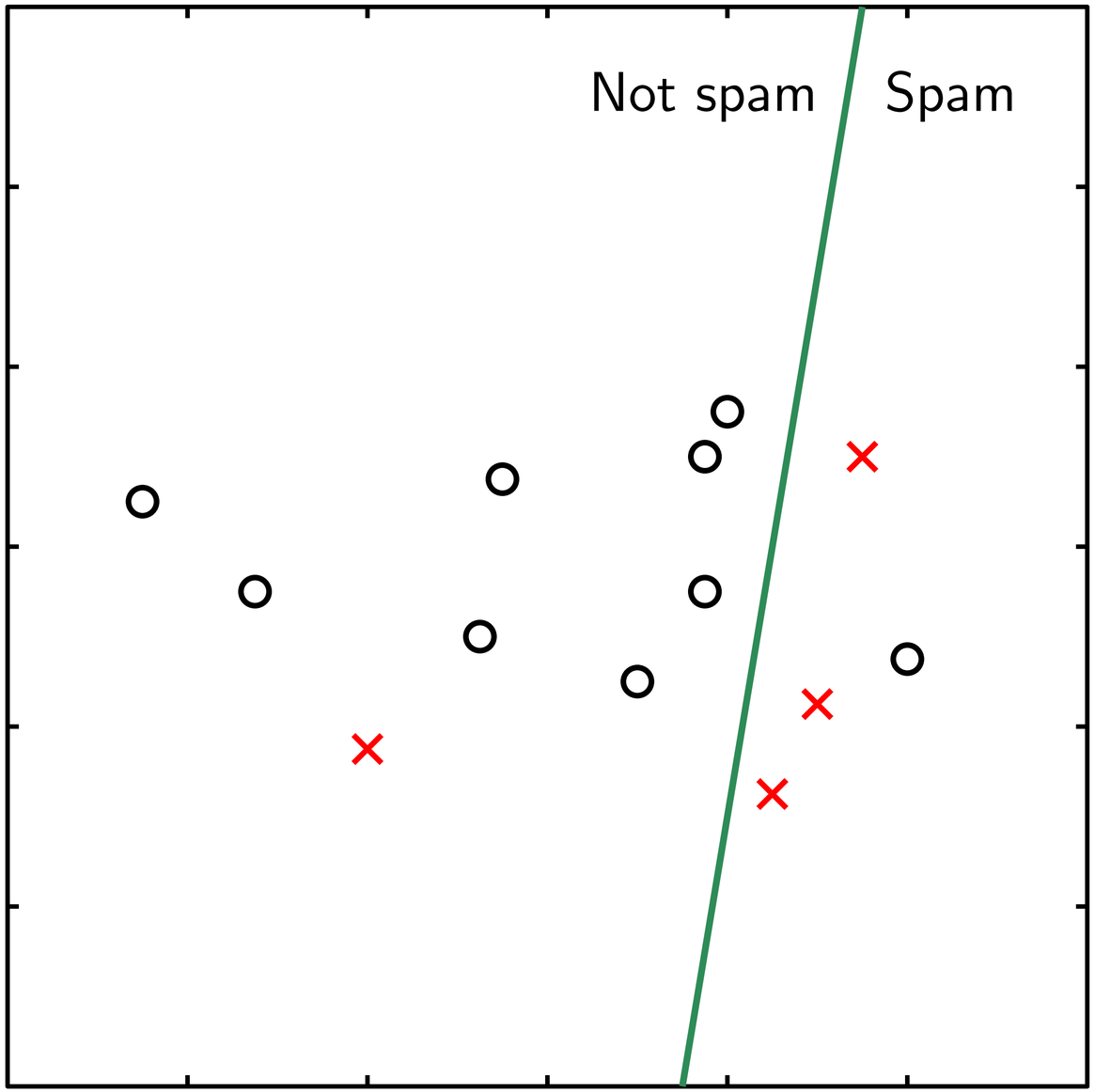
How does it work?

- Let’s say we can describe our inputs (mails) using two numbers
- We want to come up with a way to separate the circles from the crosses
- We slowly adjust our “separation rule” to correct for mistakes



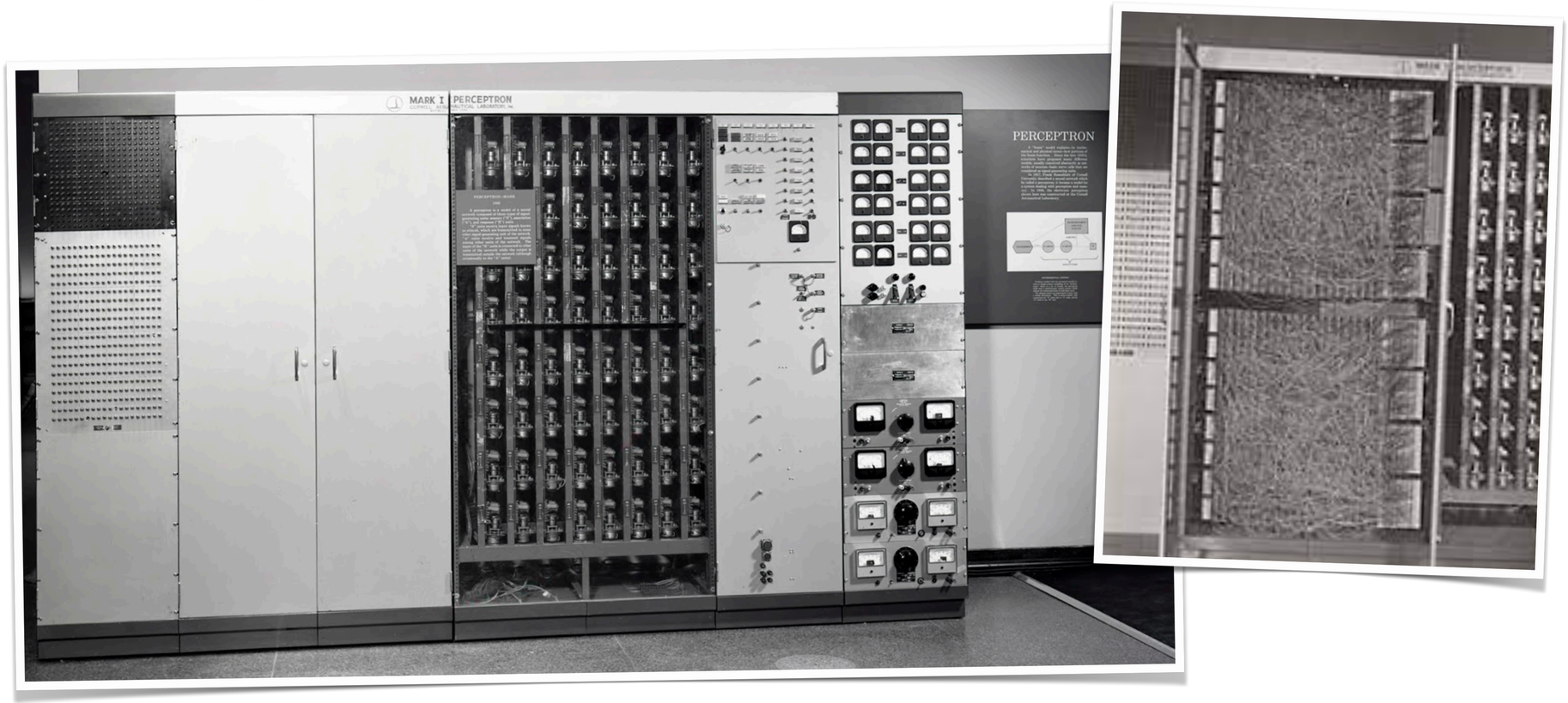
How does it work?

- Let’s say we can describe our inputs (mails) using two numbers
- We want to come up with a way to separate the circles from the crosses
- We slowly adjust our “separation rule” to correct for mistakes



The perceptron

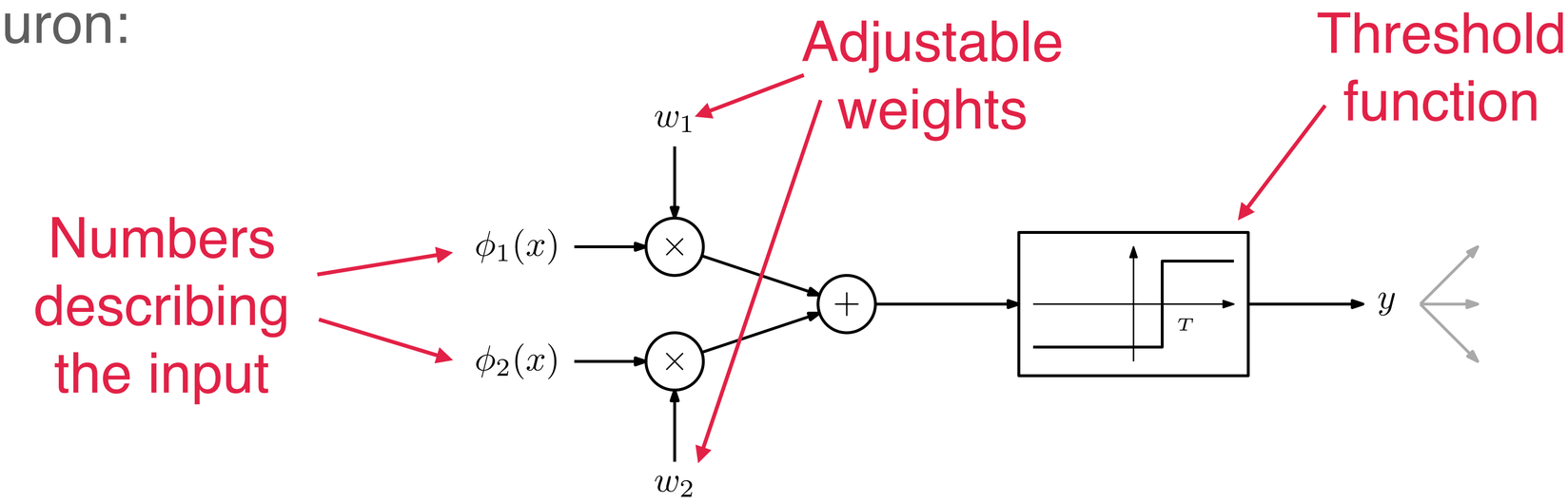
- The example just described illustrates the perceptron algorithm, a well-known learning algorithm dating back to the early days of neural networks



Mark I Perceptron

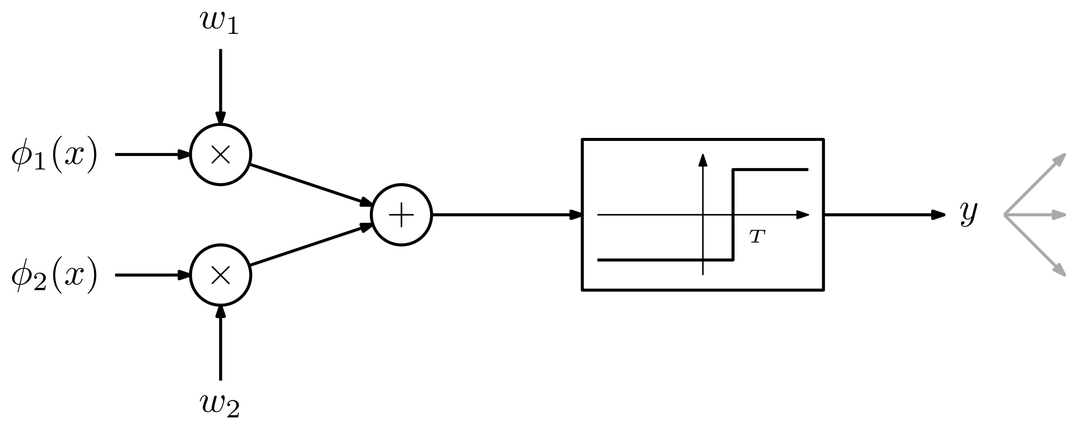
The perceptron

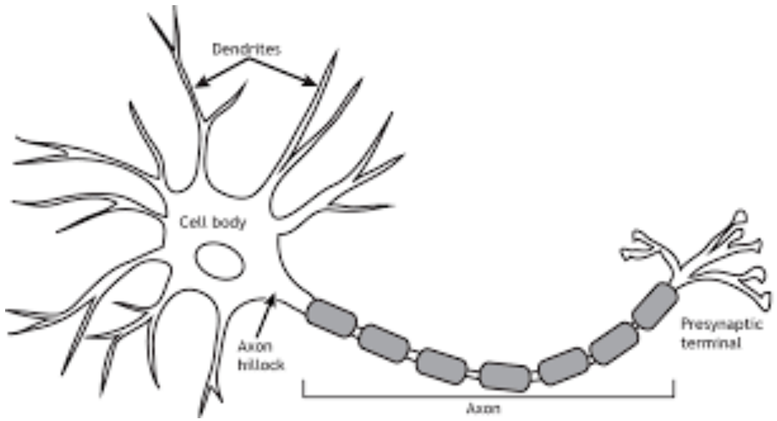
- The example just described illustrates the perceptron algorithm, a well-known learning algorithm dating back to the early days of neural networks
- The perceptron algorithm was used to train the “perceptron”, a mathematical model of a neuron:



The perceptron

- The example just described illustrates the perceptron algorithm, a well-known learning algorithm dating back to the early days of neural networks
- The perceptron algorithm was used to train the “perceptron”, a mathematical model of a neuron:





The perceptron

- The example just described illustrates the perceptron algorithm, a well-known learning algorithm dating back to the early days of neural networks
- The perceptron algorithm was used to train the “perceptron”, a mathematical model of a neuron
- When the data is linearly separable (i.e., can be perfectly split by a line/plane), the perceptron algorithm finishes in a finite number of steps

The perceptron

- The perceptron prompted a lot of excitement about the possibilities of NNs

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

1958 New York Times...

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

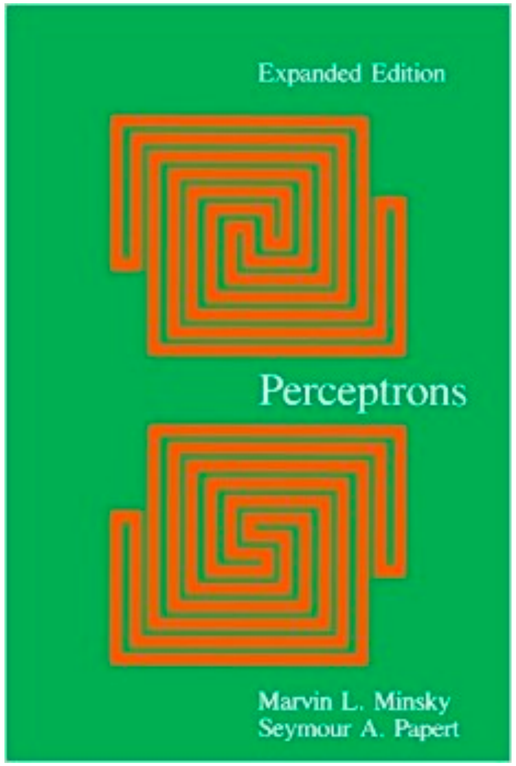
In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

The perceptron

- The perceptron prompted a lot of excitement about the possibilities of NNs
- However…



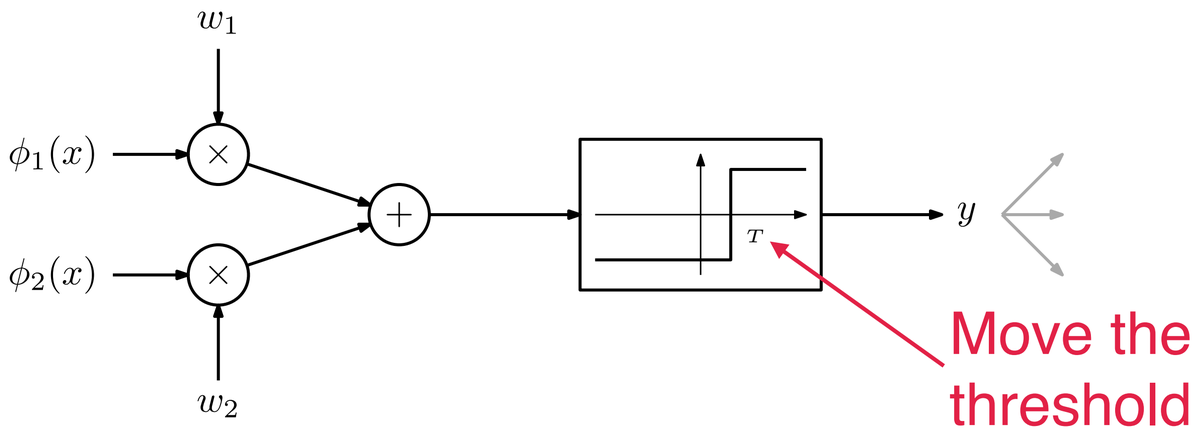
The perceptron

- The perceptron prompted a lot of excitement about the possibilities of NNs
- However…
  - In a (in-)famous book, Minsky and Papert studied the perceptron in great detail
  - The book points out several of its limitations
  - This brought about the first “neural network winter”

Let’s see what we can do, starting from the perceptron

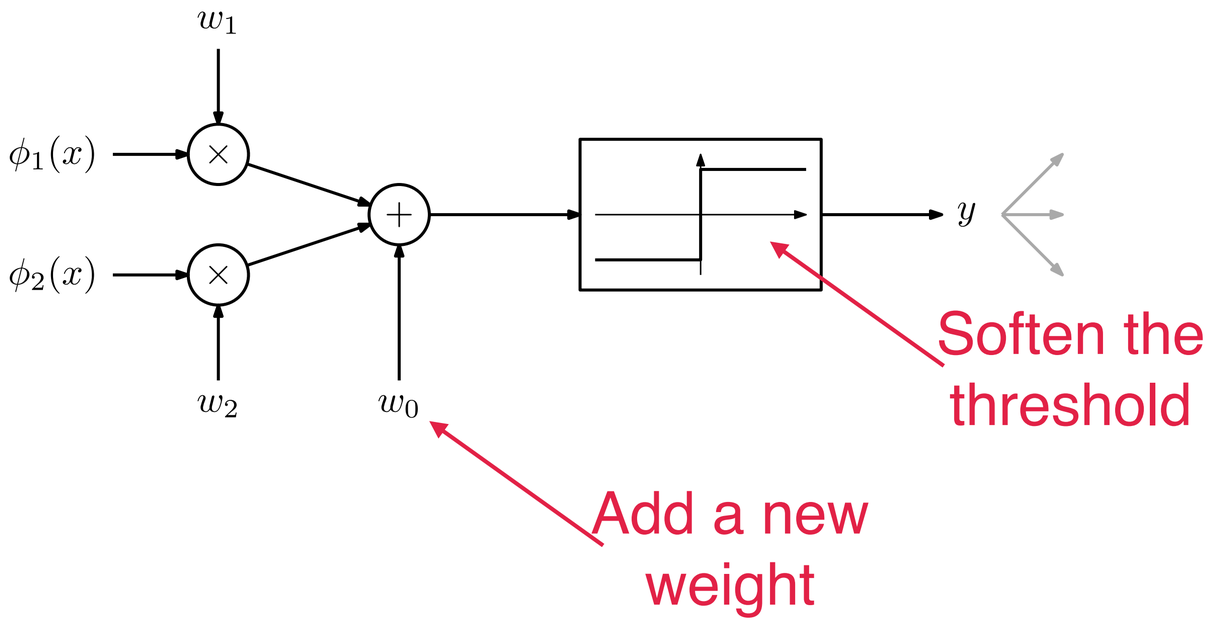
1. Get rid of the threshold

- We can re-design the perceptron model:



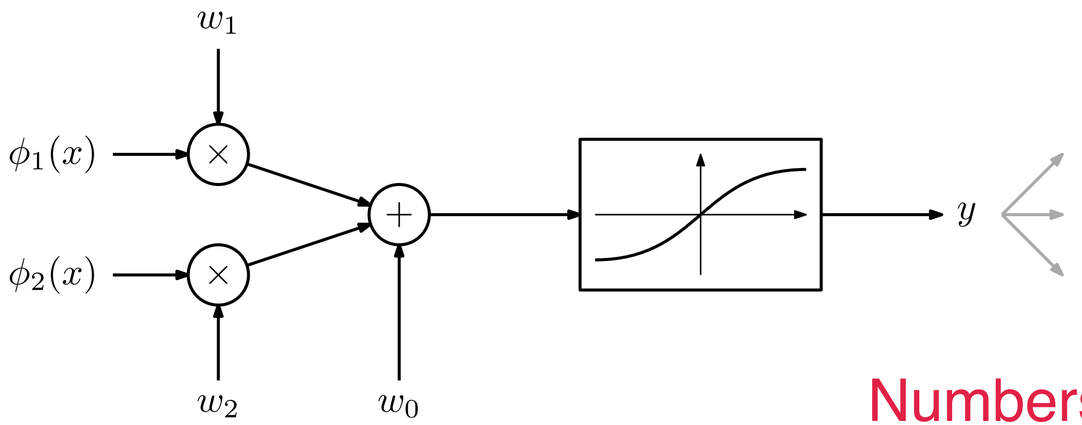
1. Get rid of the threshold

- We can re-design the perceptron model:



1. Get rid of the threshold

- We can re-design the perceptron model:



- The new model can be written as

$$y = \sigma(w^T \phi(x))$$

Numbers describing the input (vector)

Soft threshold function

Weights (also a vector)

Logistic regression

- When σ is the logistic function,

$$\sigma(z) = \frac{1}{1 + \exp(-z)},$$

  the resulting model is equivalent to the logistic regression model

- The output of the model can be interpreted as a probability (e.g., the probability that *x* is spam)

2. Going beyond spam

- In the spam filter problem, we have only two possible outputs (spam/no spam)
  - It is a binary classification problem
- What if we have multiple (more than two) discrete outputs?
- What if we have continuous outputs?

Multiple discrete outputs

- When we have multiple discrete outputs, we have a multiclass (or multinomial) classification problem
- We can use the same general idea
  - We assign to each possible output (class) a score:

$$z_i = w_i^T \phi(x)$$

Score for class *i*

Weights for class *i*

Multiple discrete outputs

- The scores can be turned into probabilities using the softmax function:

$$y_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} = \mathbb{P}[\text{class} = i \mid x]$$

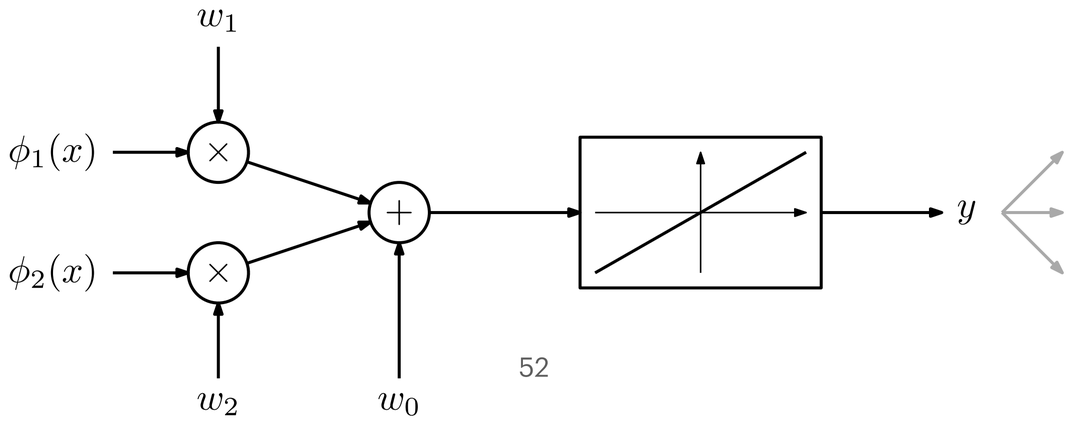
- When the number of classes is 2, the model is equivalent to the logistic regression model
  - … but the latter has only half the parameters

Continuous outputs

- When we have continuous outputs, we have a regression problem
- We predict the (continuous) output to be

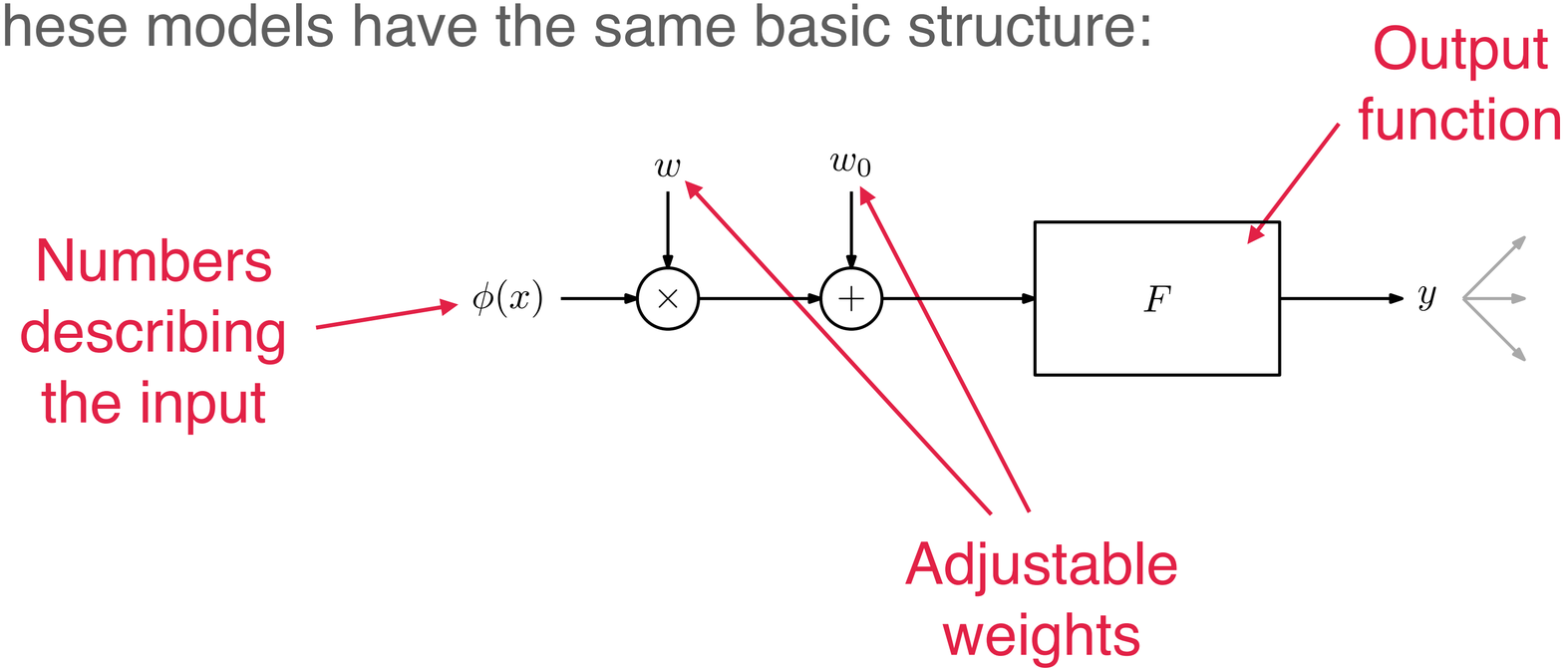
$$y = w^T \phi(x)$$

which corresponds to the model



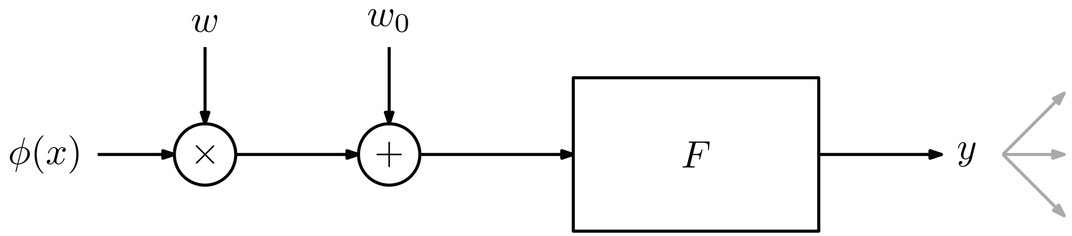
Linear models

- All these models have the same basic structure:



Linear models

- All these models have the same basic structure:



- They are generally referred as linear models

3. Training

- In the perceptron algorithm, “learning” (training) minimized the number of mistakes
- In general, we will want to minimize the risk of our model,

$$\mathbb{E}_{(x,y)\sim\mu}[L(x, y; w)]$$

where *L* is some loss function

Distribution governing input-output (unknown)

3. Training

- Since we can't minimize the risk (we don't know μ), we instead use the empirical risk,

$$\mathbb{E}_{(x,y)\sim\mu}[L(x,y;w)] \approx \boxed{\frac{1}{N}\sum_{n=1}^{N} L(x_n,y_n;w)}$$

Empirical risk
(risk in our dataset)

3. Training

- Since we can't minimize the risk (we don't know μ), we instead use the empirical risk,

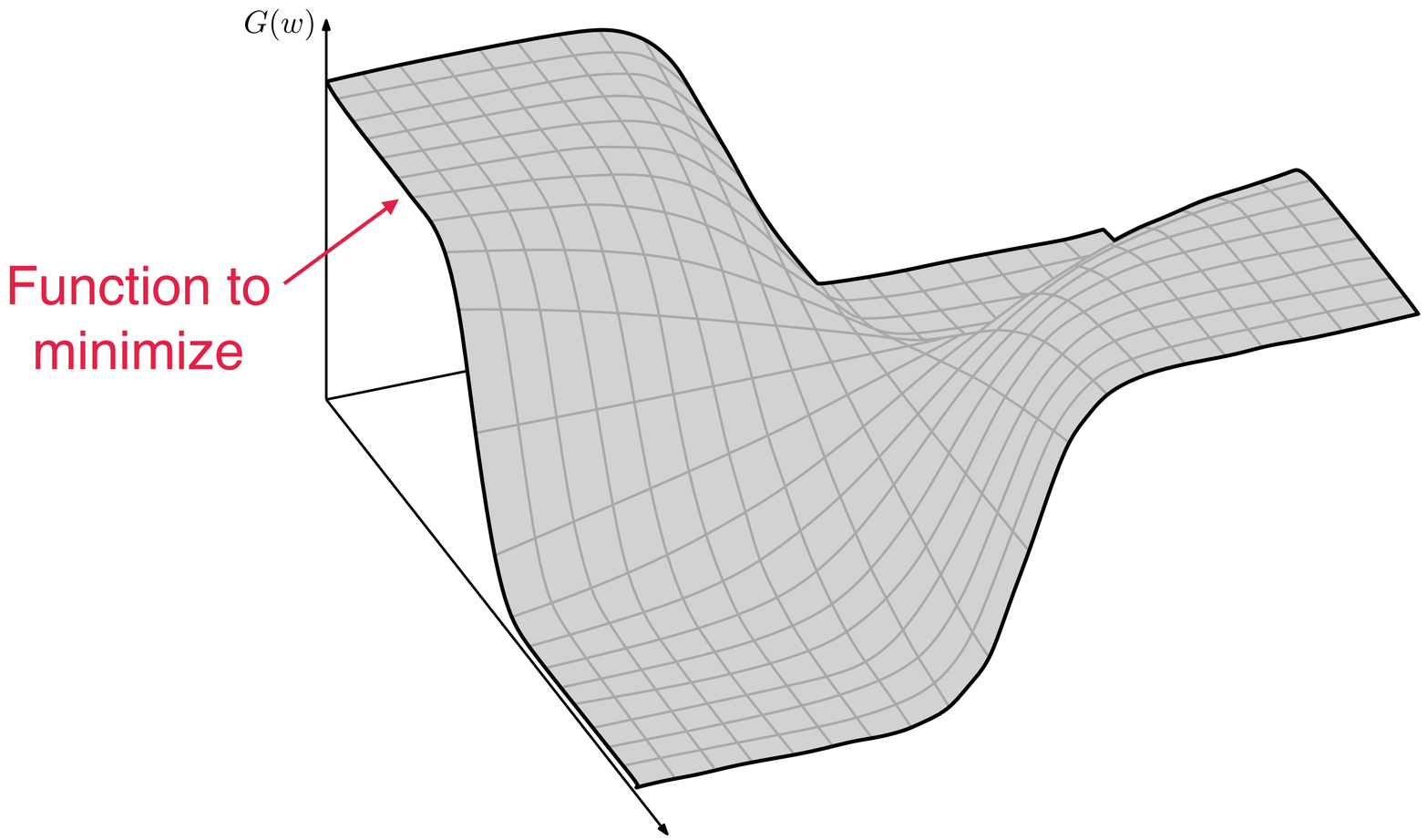
$$\mathbb{E}_{(x,y)\sim\mu}[L(x,y;w)] \approx \frac{1}{N}\sum_{n=1}^{N} L(x_n,y_n;w)$$

- If the loss is adequately chosen, we can optimize W iteratively using gradient descent on the empirical risk,

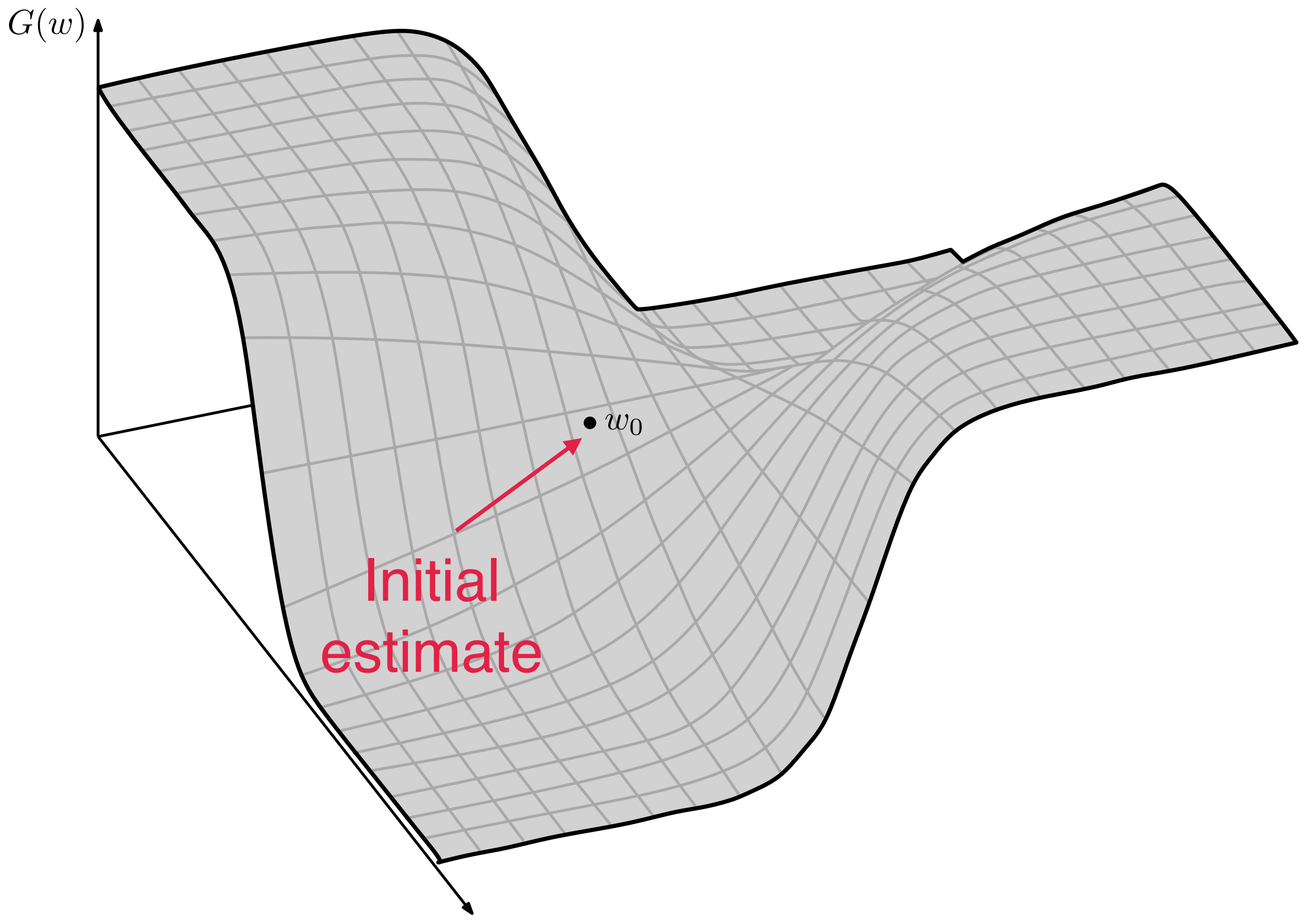
$$w^{(k+1)} \leftarrow w^{(k)} - \alpha \sum_{n=1}^{N} \nabla_w L(x_n,y_n;w^{(k)})$$

Step-size or learning rate

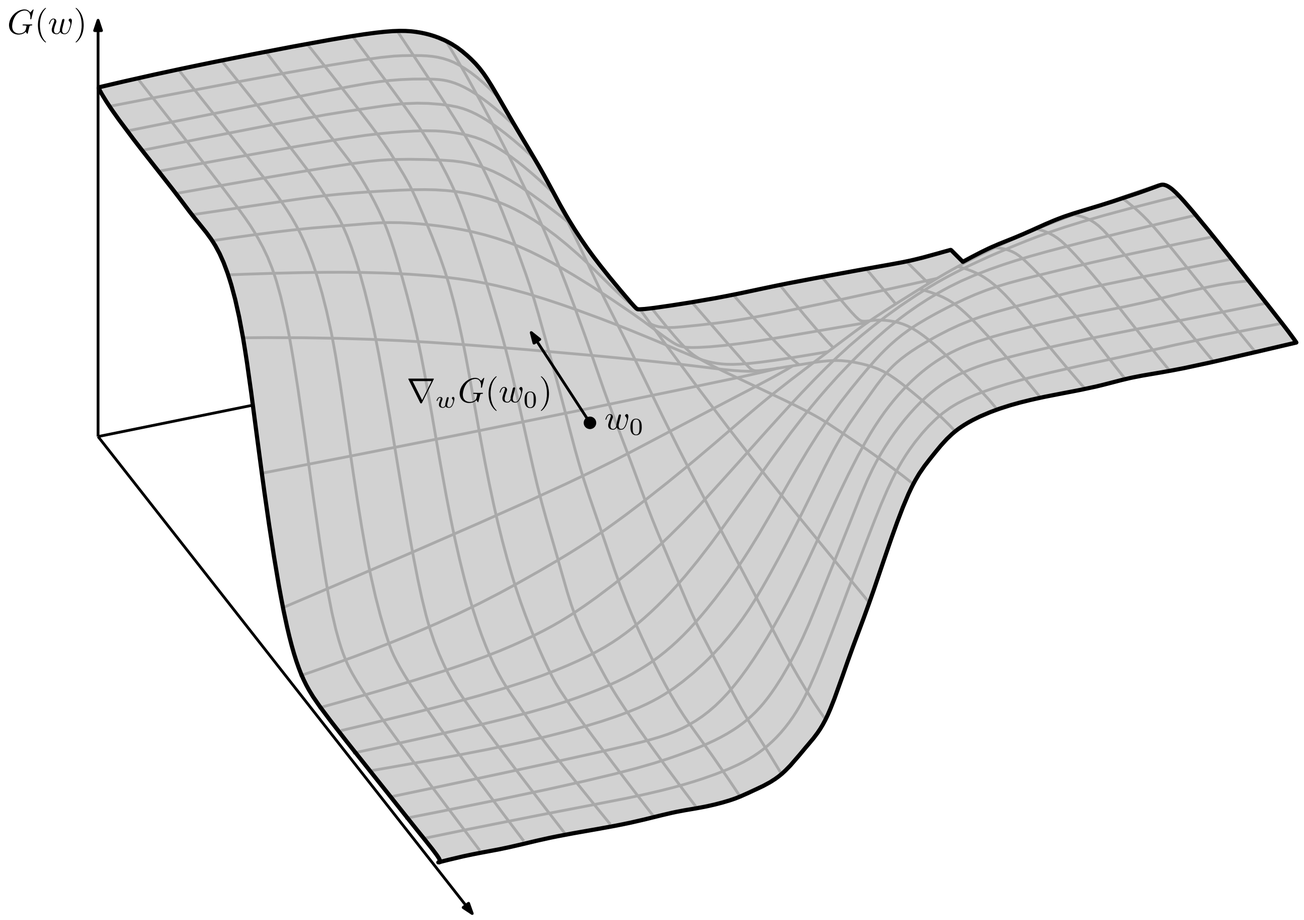
(Stochastic) Gradient descent



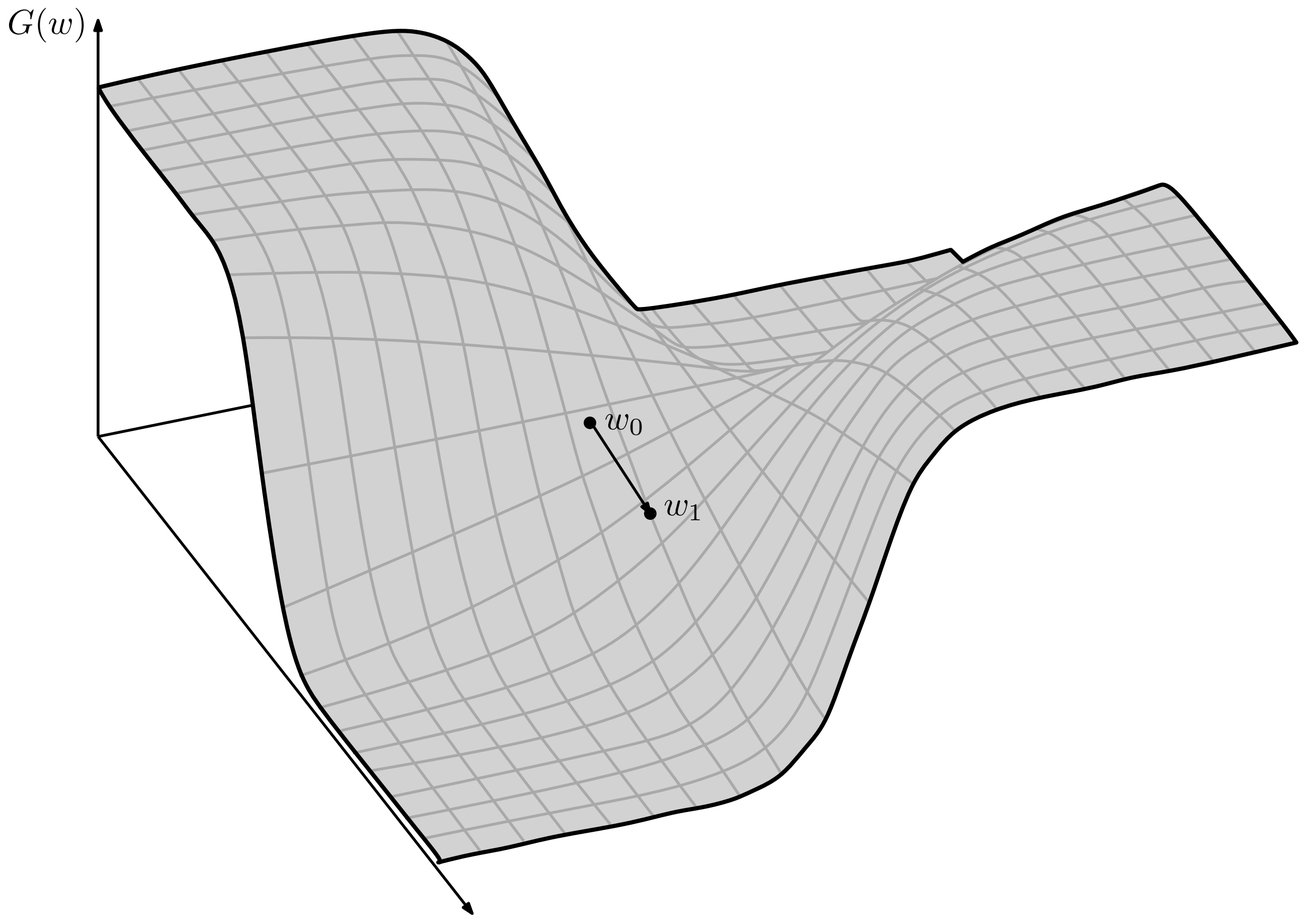
(Stochastic) Gradient descent



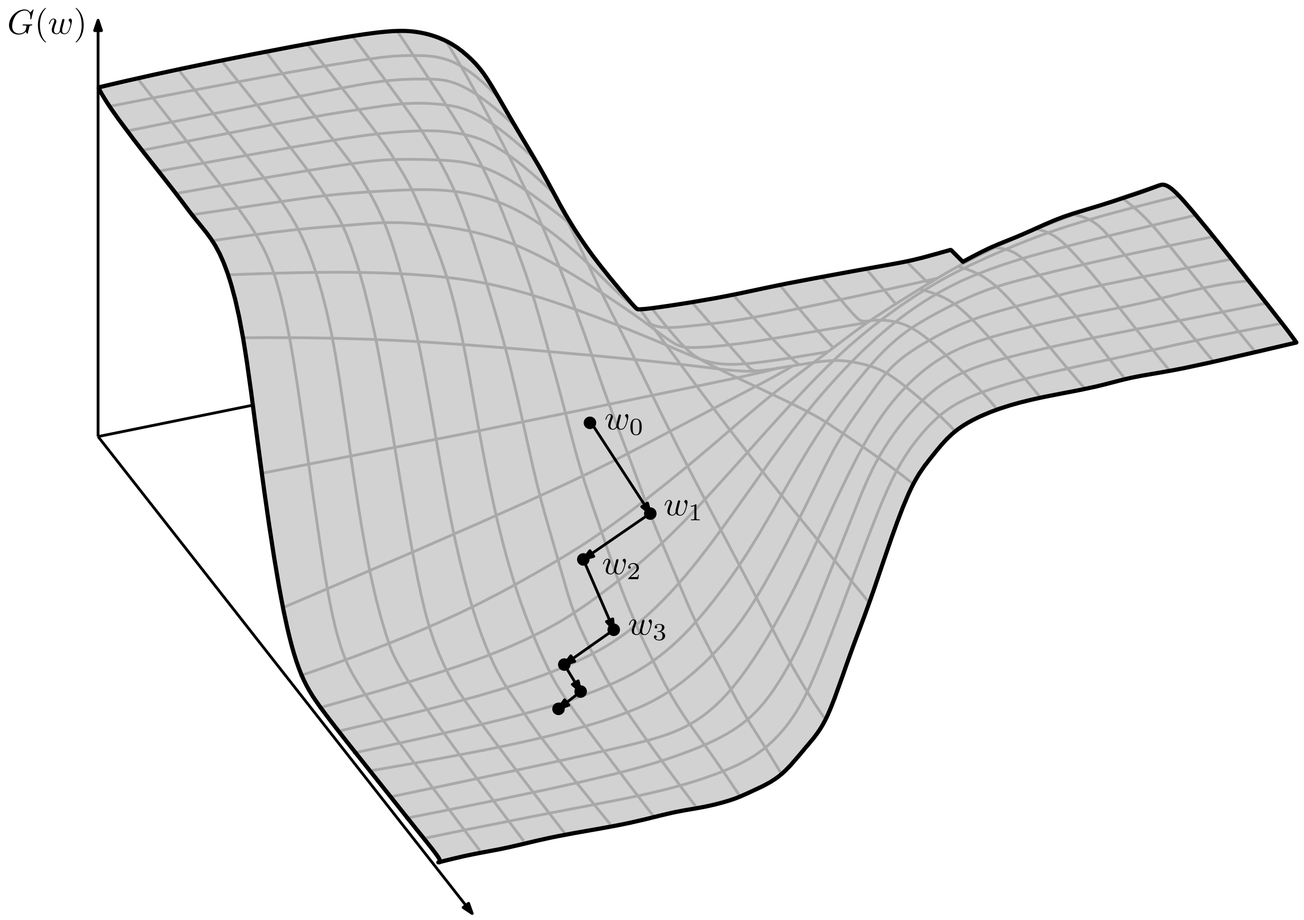
(Stochastic) Gradient descent



(Stochastic) Gradient descent



(Stochastic) Gradient descent



(Stochastic) Gradient descent

- For large datasets, the gradient

$$\sum_{n=1}^{N} \nabla_w L(x_n, y_n; w^{(k)})$$

  may take too long to compute

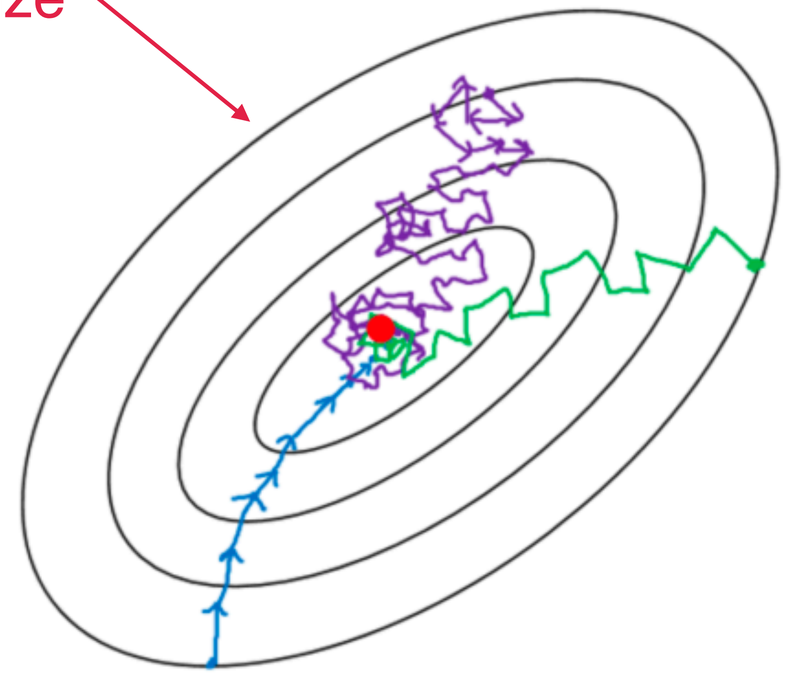
- Instead, we can use mini-batches of samples or even a single sample:

$$w^{(k+1)} \leftarrow w^{(k)} - \alpha \nabla_w L(x_n, y_n; w^{(k)})$$

Stochastic gradient descent

(Stochastic) Gradient descent

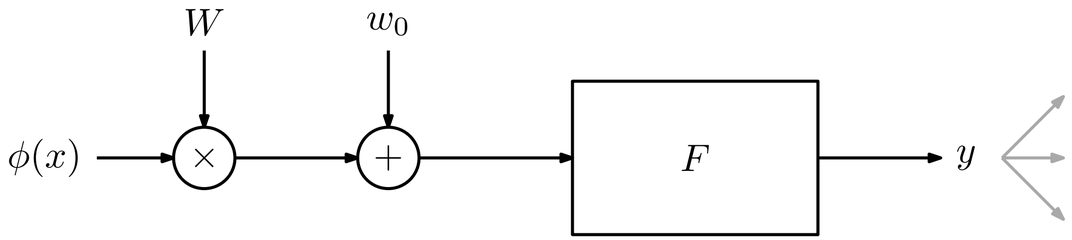
Level lines of function to minimize



Batch gradient descent
Mini-batch gradient Descent
Stochastic gradient descent

Putting everything together…

- Task: Learn a model



- Data: A set of input-output pairs

$$\mathcal{D} = \{(x_n, y_n), n = 1, \ldots, N\}$$

- Performance: The risk

$$\mathbb{E}_{(x,y)\sim\mu}[L(x, y; w)]$$

Putting everything together…

- Training: Use (stochastic) gradient descent to minimize the empirical risk

$$\frac{1}{N}\sum_{n=1}^{N} L(x_n, y_n; w)$$

… but…

… we have **three** problems left

Problem 1.

What we are optimizing is not what we want to optimize

- We want to optimize the risk,

$$\mathbb{E}_{(x,y)\sim\mu}[L(x, y; w)]$$

- We are optimizing a surrogate—the empirical risk,

$$\frac{1}{N}\sum_{n=1}^{N} L(x_n, y_n; w)$$

Inductive learning hypothesis: If we do well in the training data, we will also do well at “test time”.

Problem 1.

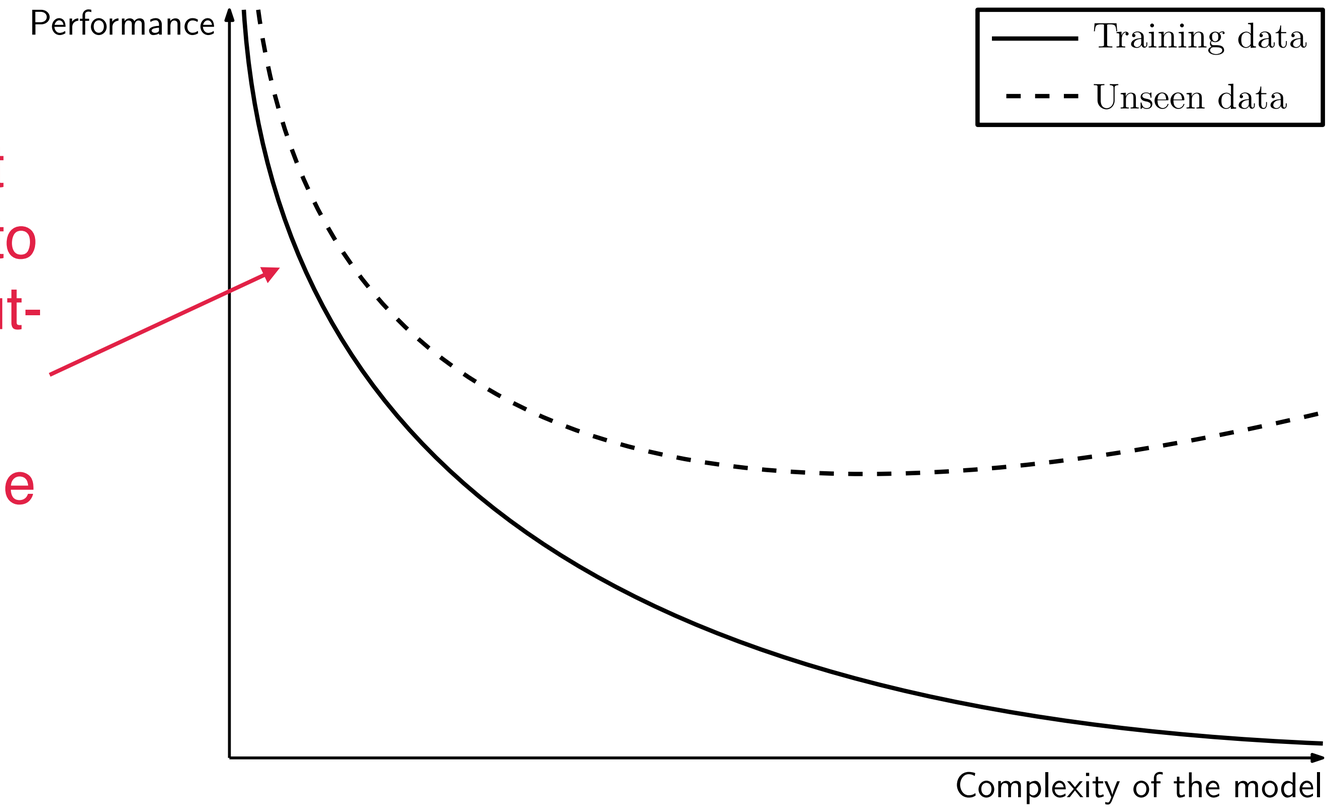
What we are optimizing is not what we want to optimize

- Unfortunately, this is not necessarily the case

Underfitting

The model is not powerful enough to represent the input-output relation

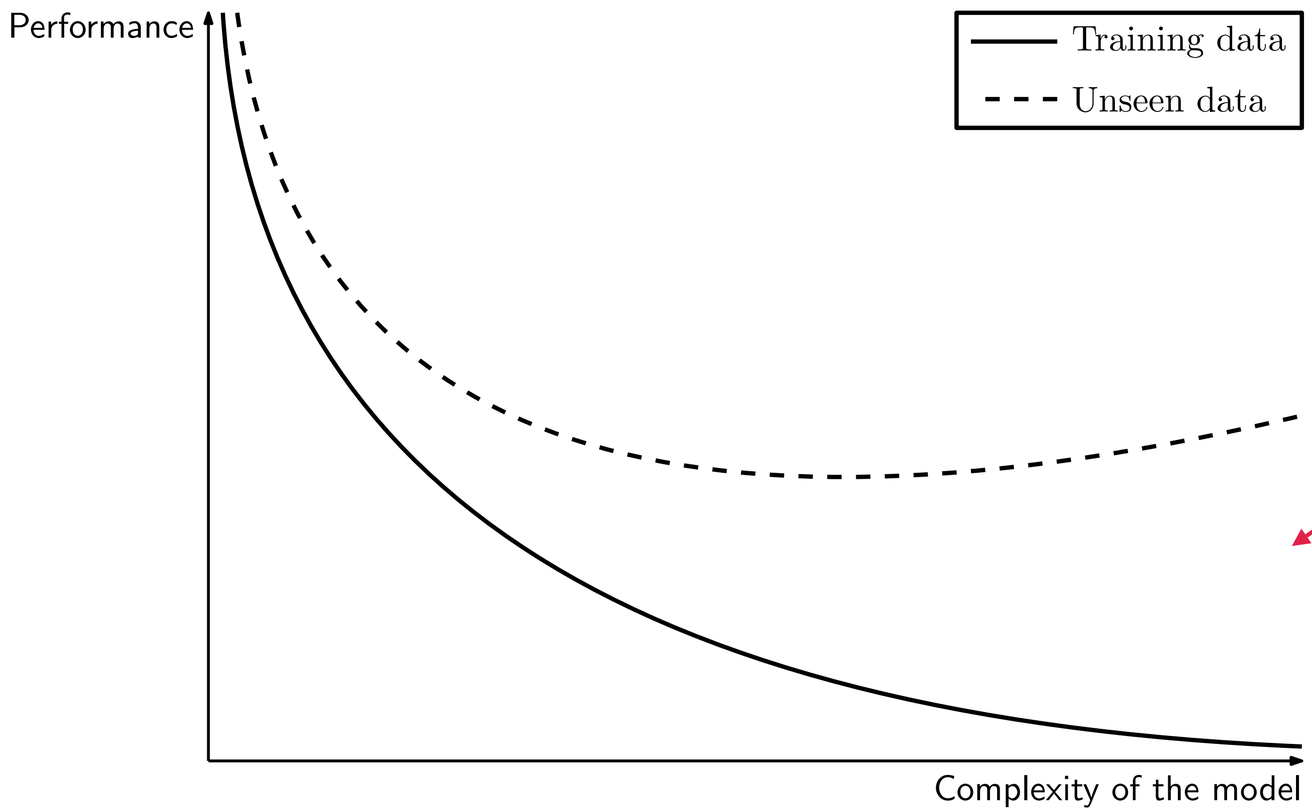
Performance in the data is similar to "real-world" performance



Problem 1.

What we are optimizing is not what we want to optimize

- Unfortunately, this is not necessarily the case



Overfitting

The model is so powerful that it learns spurions input-output relations in data

Performance in the data overestimates "real-world" performance

Problem 1.

What we are optimizing is not what we want to optimize

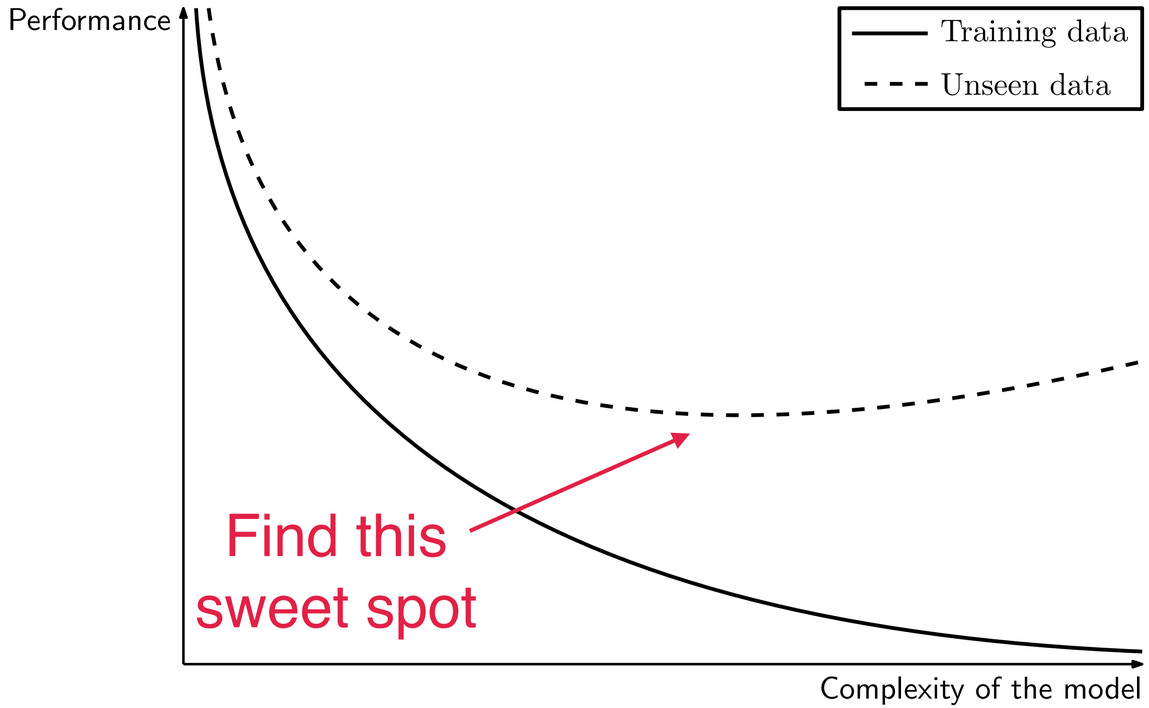
- Unfortunately, this is not necessarily the case
- Overfitting is a serious problem in machine learning
  - Model is over-sensitive to data—small change in training data causes big change in the resulting model
  - Model appears to be performing well during training, but performs bad after deployment

Overfitting

- How to spot overfitting?
  - Overfitting apparent by comparing performance in the training data with performance with unseen data
  - We split the data and use only part of it in training

Overfitting

- How to tackle overfitting?
  - Solution 1: Use simpler model (trial and error)



Overfitting

- How to tackle overfitting?
  - Solution 2: Allow complex models, drive learning to prefer simpler models
  - Replace empirical risk by regularized empirical risk

$$\frac{1}{N}\sum_{n=1}^{N} L(x_n, y_n; w)$$

Overfitting

- How to tackle overfitting?
  - Solution 2: Allow complex models, drive learning to prefer simpler models
  - Replace empirical risk by regularized empirical risk

$$\frac{1}{N}\sum_{n=1}^{N} L(x_n, y_n; w) + \lambda R(w)$$

Penalize complex models

Regularization

- In regularized risk minimization, we are willing to sacrifice accuracy in the training data to have a “simpler” model

$$\frac{1}{N}\sum_{n=1}^{N} L(x_n, y_n; w) + \lambda R(w)$$

The parameter λ controls the tradeoff

- Example: L^2 regularization (also called ridge regression or weight decay)

$$R(w) = \frac{1}{2}\|w\|_2^2$$

Pushes *w* towards the origin

Regularization

- In regularized risk minimization, we are willing to sacrifice accuracy in the training data to have a “simpler” model

$$\frac{1}{N}\sum_{n=1}^{N} L(x_n, y_n; w) + \lambda R(w)$$

- Example: L^1 regularization (also called lasso)

$$R(w) = \|w\|_1$$

Leads to sparse *w*

Problem 2.

Choosing the loss function

- The loss function should be aligned with the goal of the designer
- Many loss functions are possible

Choosing the loss function

- The loss function should be aligned with the goal of the designer
- Example: Squared error in regression

$$L(x, y; w) = \frac{1}{2}\|y - w^T\phi(x)\|^2$$

Distance between "target value", *y*, and estimated value, $w^T\phi(x)$

Choosing the loss function

- The loss function should be aligned with the goal of the designer
- Example: Negative log-likelihood in classification

$$L(x, y; w) = -\log \mathbb{P}[\text{class} = y \mid x; w]$$

Probability of "target class", *y*

The empirical risk using a negative log-likelihood loss is equivalent to the cross-entropy between the true class distribution and the estimated class probabilities

Choosing the loss function

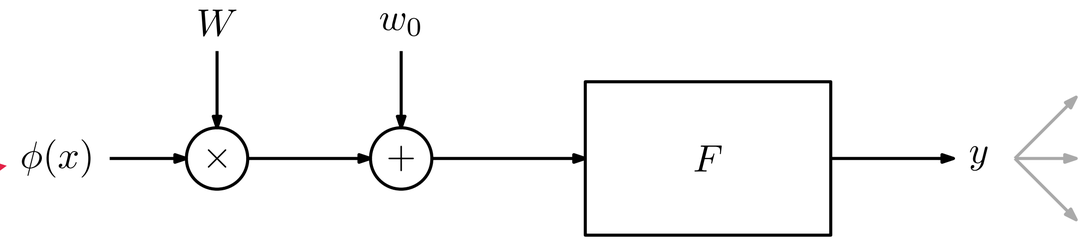
- The loss function should be aligned with the goal of the designer
- Many loss functions are possible
  - Squared loss, negative log-likelihood, etc.
- We can design losses specific for unsupervised learning, reinforcement learning, etc.

Problem 3.

Choosing the features

- Models so far adhere to the structure

Numbers describing the input

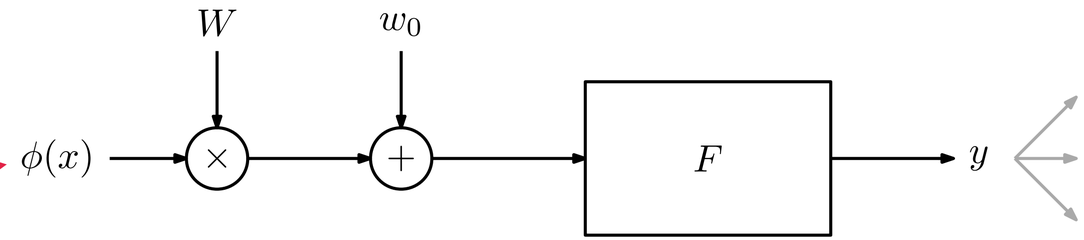


Problem 3.

Choosing the features

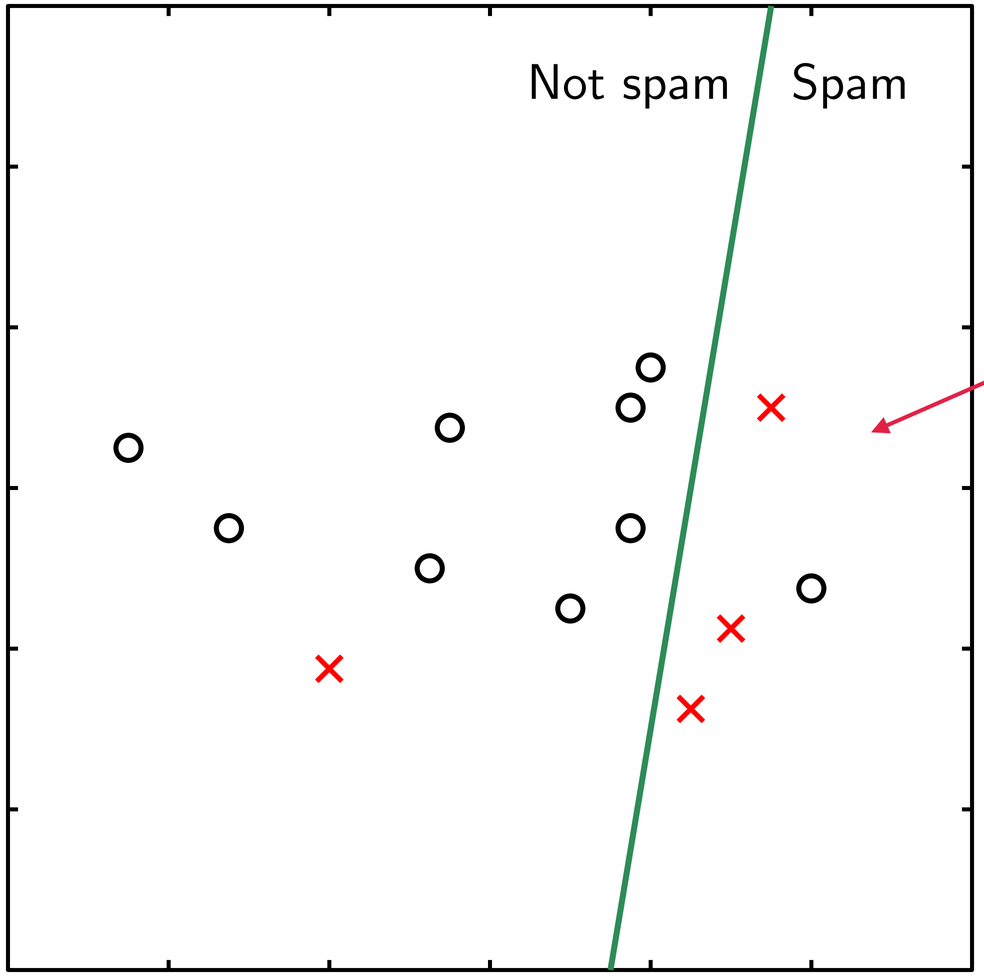
- Models so far adhere to the structure

Features describing the input



- Good features may lead to better results
- Bad features may lead to poor results

Going back to our example…

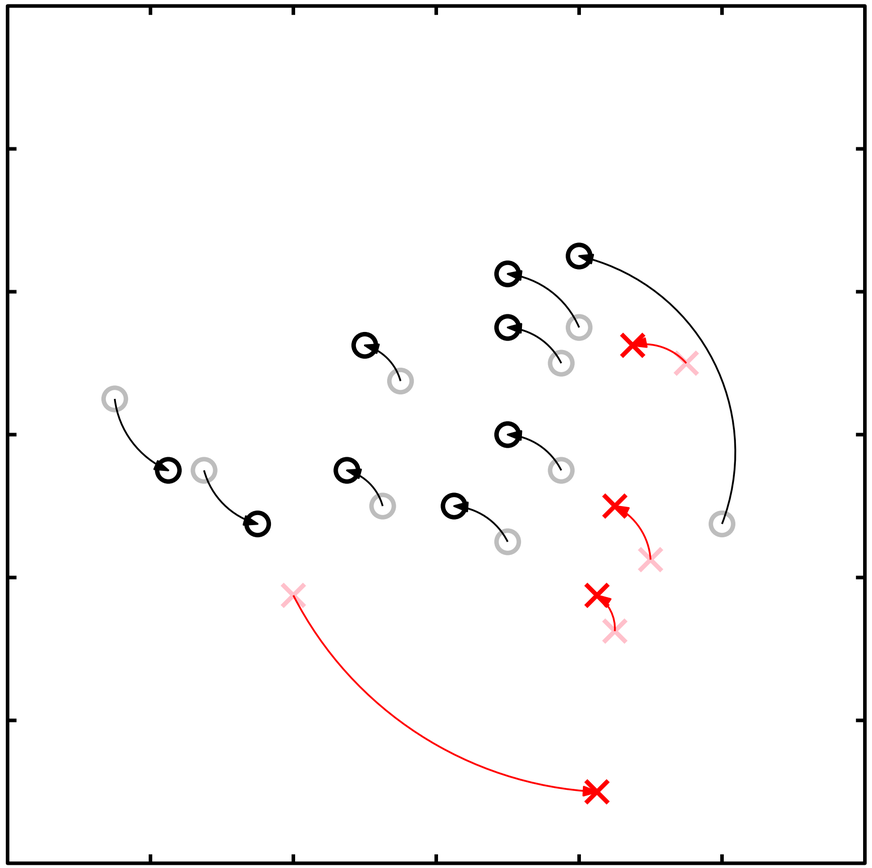


With these features, we can't perfectly separate the two classes

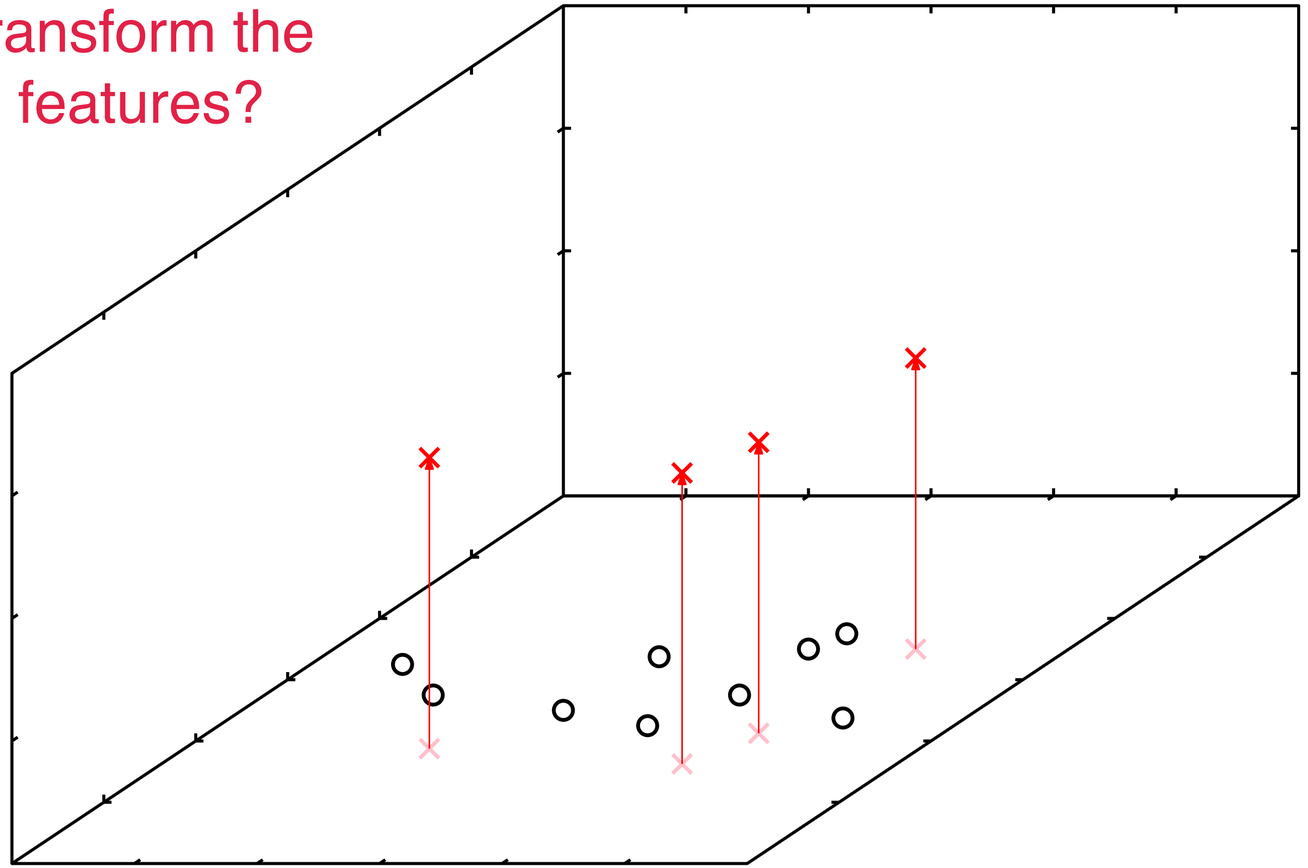
Going back to our example…

- With the current representation (features)…
  - … we are unable to perfectly classify the data
  - … the data is not linearly separable

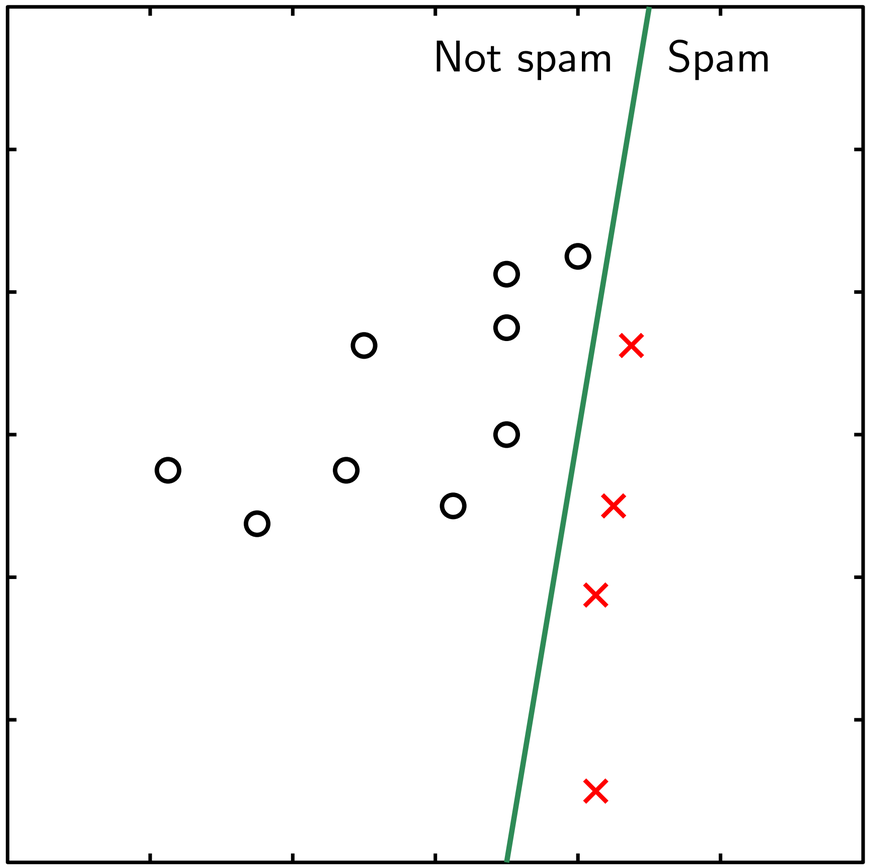
Going back to our example…



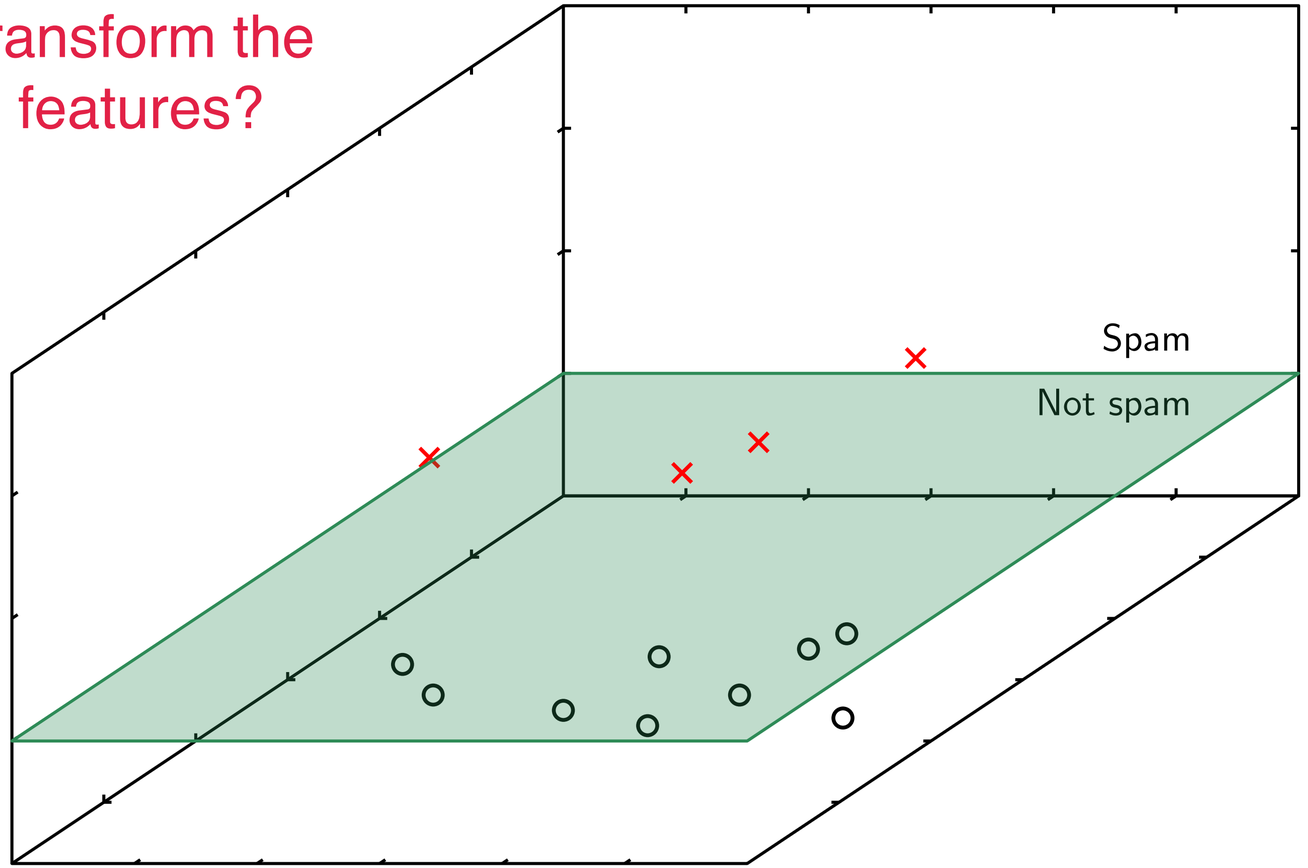
What if we transform the features?



Going back to our example…

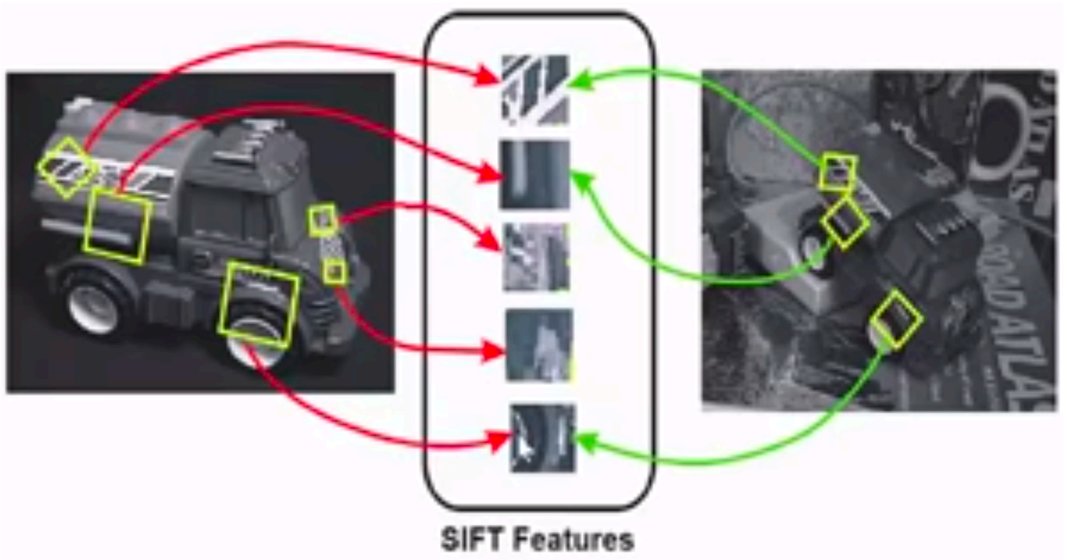


What if we transform the features?



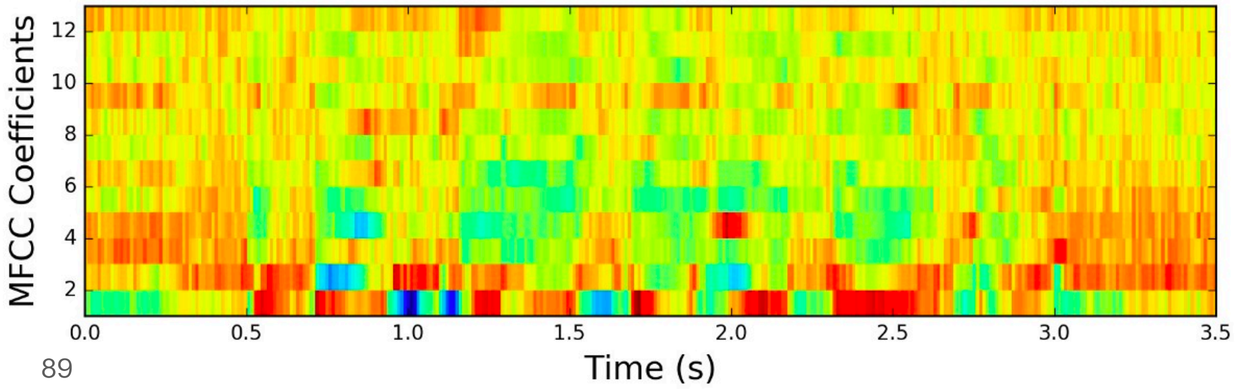
Choosing the features…

- Feature design and selection was a core problem in “classical” machine learning
- Significant effort went into designing “good” features
- E.g.: SIFT features for computer vision



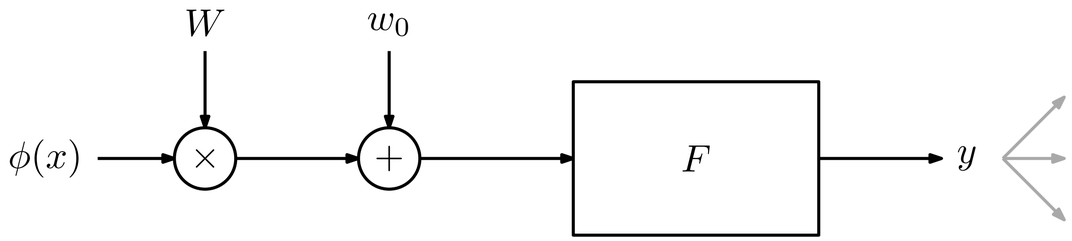
Choosing the features…

- Feature design and selection was a core problem in “classical” machine learning
- Significant effort went into designing “good” features
- E.g.: MFCC features for speech processing



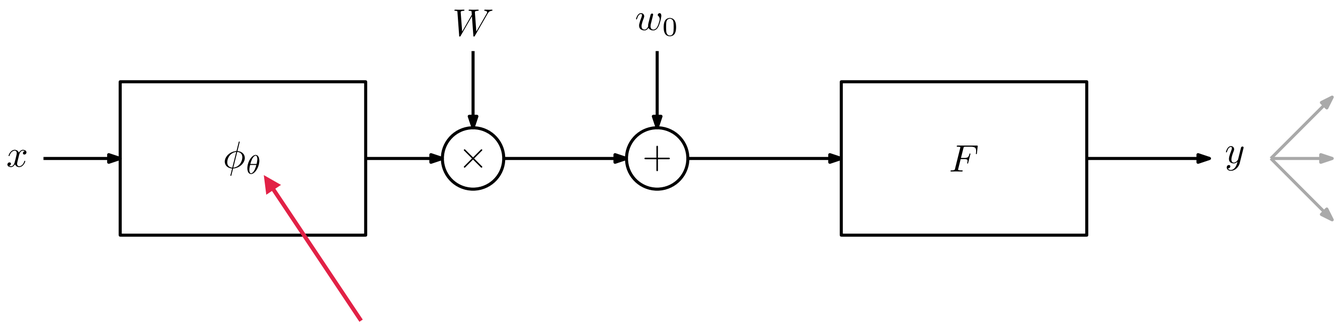
Neural networks to the rescue

- Neural networks alleviate the need to hand-designed features



Neural networks to the rescue

- Neural networks alleviate the need to hand-designed features



The features themselves are learned

Deep learning models automatically learn representations for the data

What is Deep Learning?

- Neural networks

- Anything beyond shallow (linear) models for machine learning

- Anything that learns representations

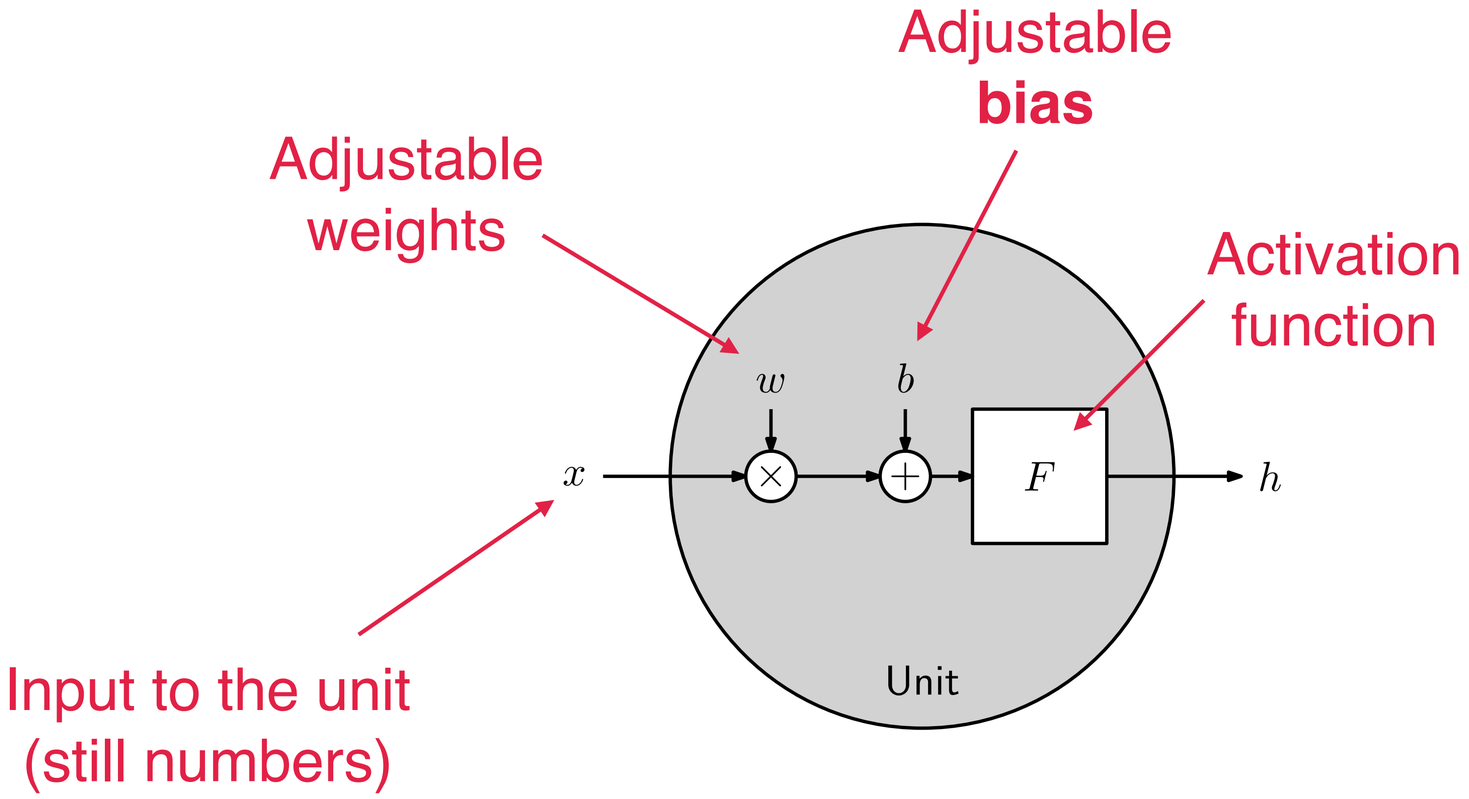
Learning with neural networks

What does a neural network look like?

- Informally, a neural network is a “mesh” of neurons (often called units)
  - Each unit is a perceptron-like element
- Units in a neural network are arranged in layers
  - We sometimes refer to the number of layers as the depth of the network
  - We sometimes refer to the number of units per layer as the width of the network

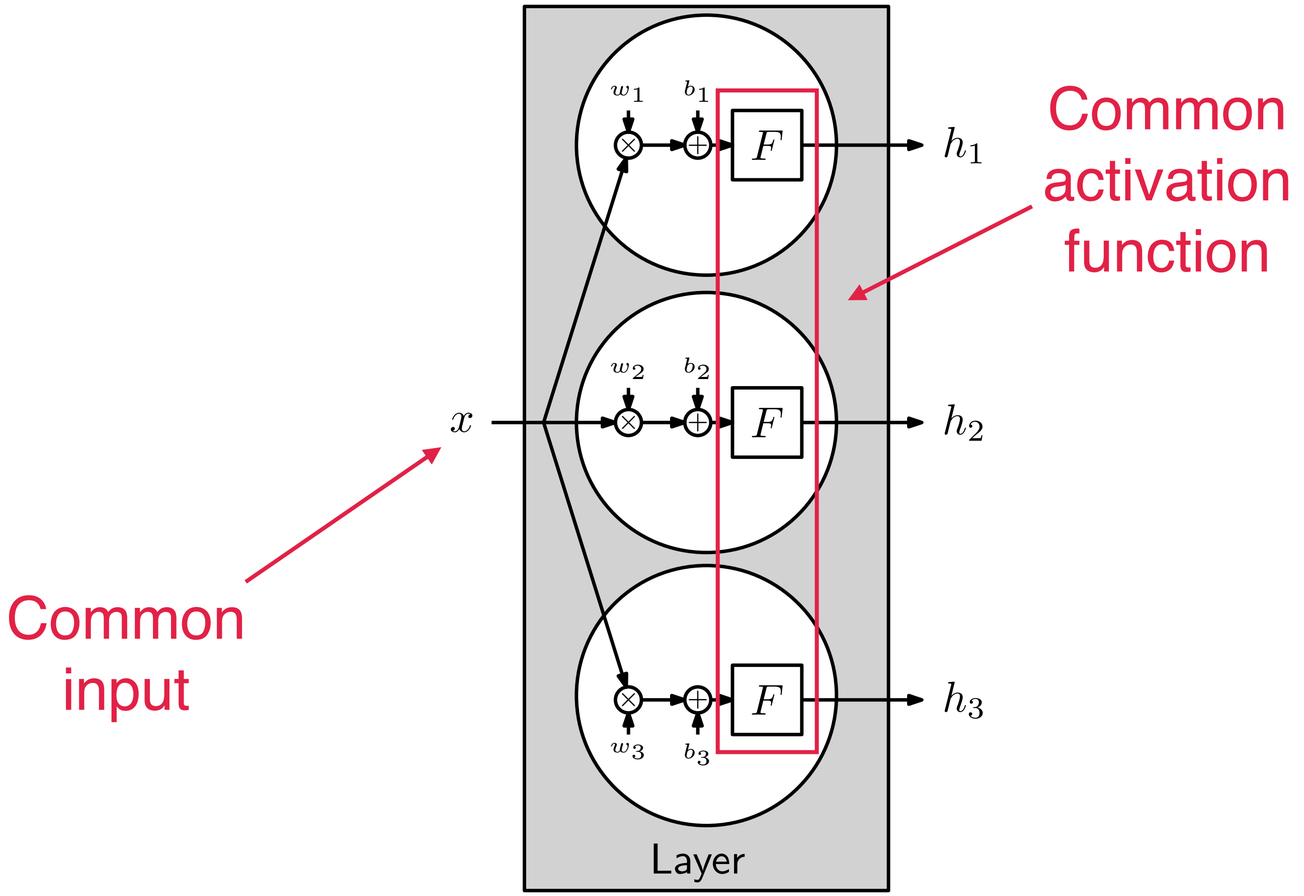
What does a neural network look like?

Unit



What does a neural network look like?

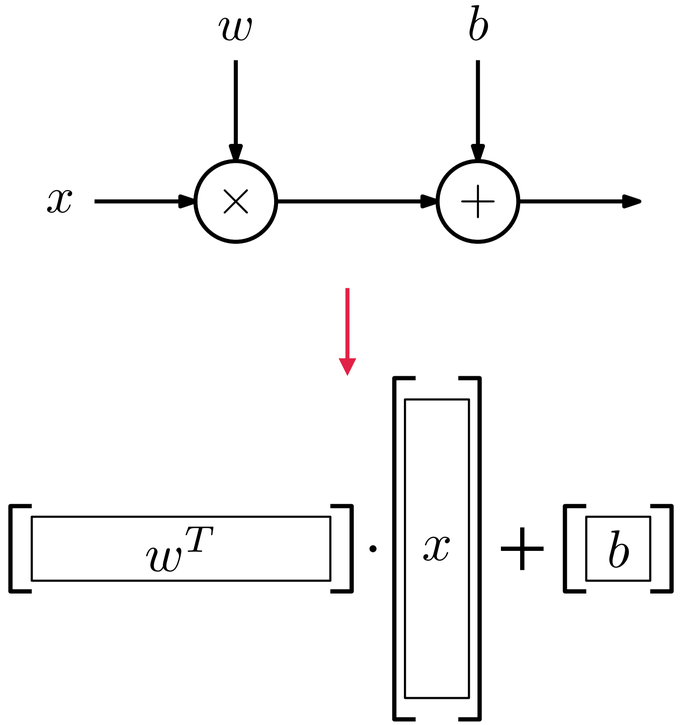
Layer



What does a neural network look like?

Layer

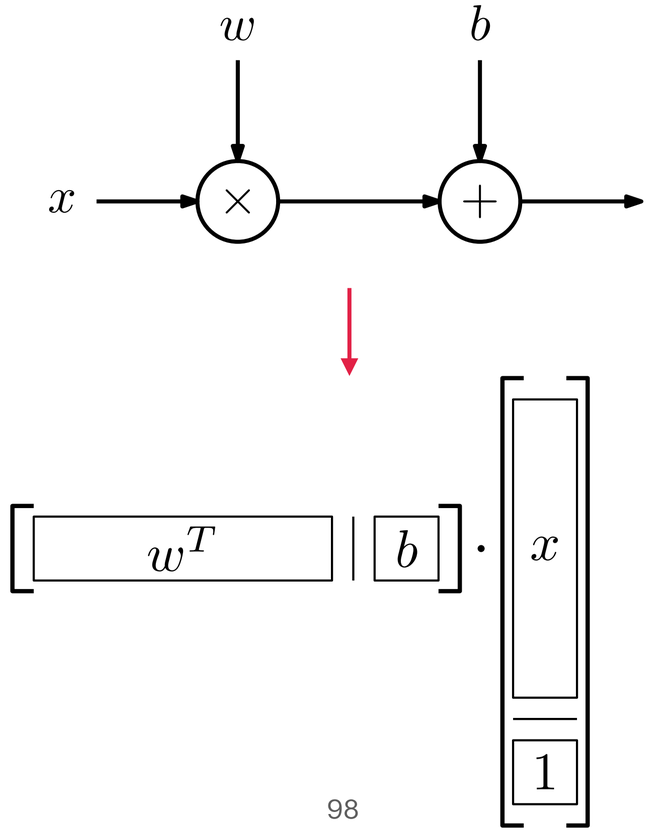
- We can move the bias into the weight vector:



What does a neural network look like?

Layer

- We can move the bias into the weight vector:

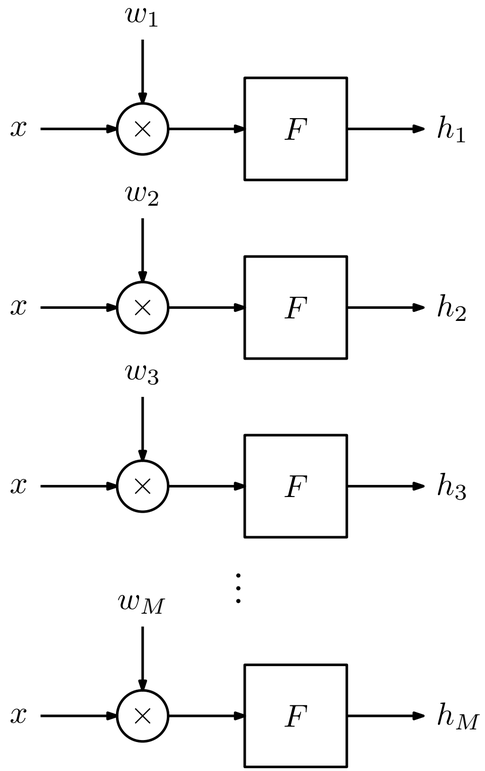


For simplicity, we often write only $w^T x$ (b is assumed in w)

What does a neural network look like?

Layer

- We can further compress the notation:



$$h_1 = F\left(\begin{bmatrix} w_1^T \end{bmatrix} \cdot \begin{bmatrix} x \end{bmatrix}\right)$$

$$h_2 = F\left(\begin{bmatrix} w_2^T \end{bmatrix} \cdot \begin{bmatrix} x \end{bmatrix}\right)$$

$$h_3 = F\left(\begin{bmatrix} w_3^T \end{bmatrix} \cdot \begin{bmatrix} x \end{bmatrix}\right)$$

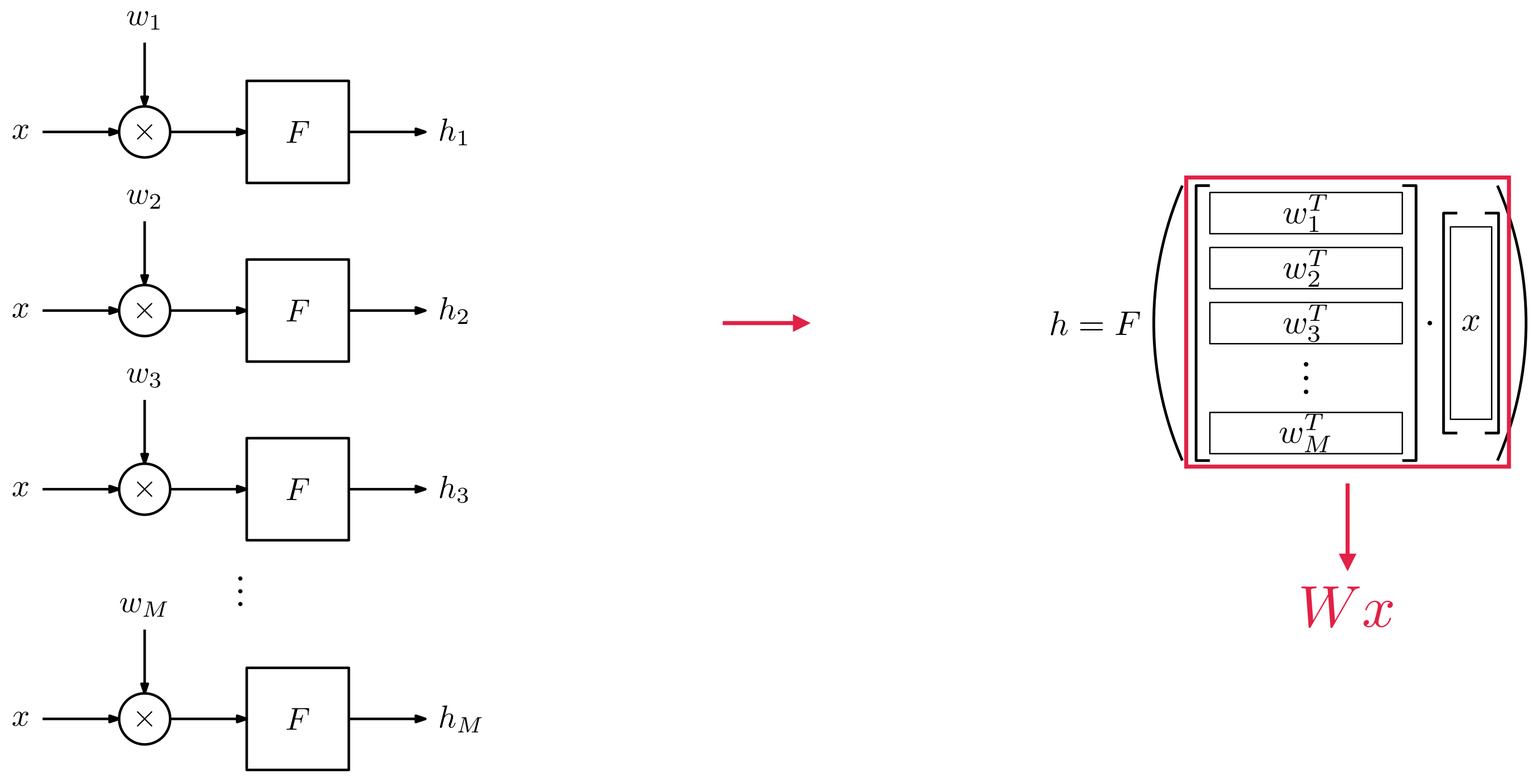
$$\vdots$$

$$h_M = F\left(\begin{bmatrix} w_M^T \end{bmatrix} \cdot \begin{bmatrix} x \end{bmatrix}\right)$$

What does a neural network look like?

Layer

- We can further compress the notation:



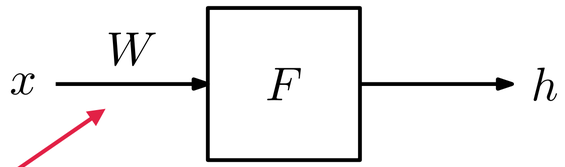
What does a neural network look like?

Layer

- In vector notation,

$$h = F(Wx)$$

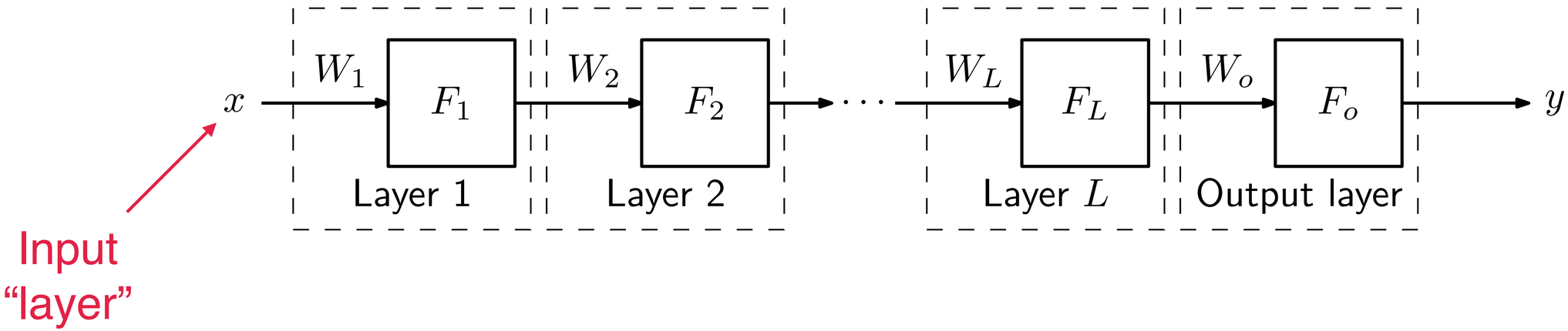
- In diagram form,



Input *x* is multiplied by *W* and "passed through" *F*

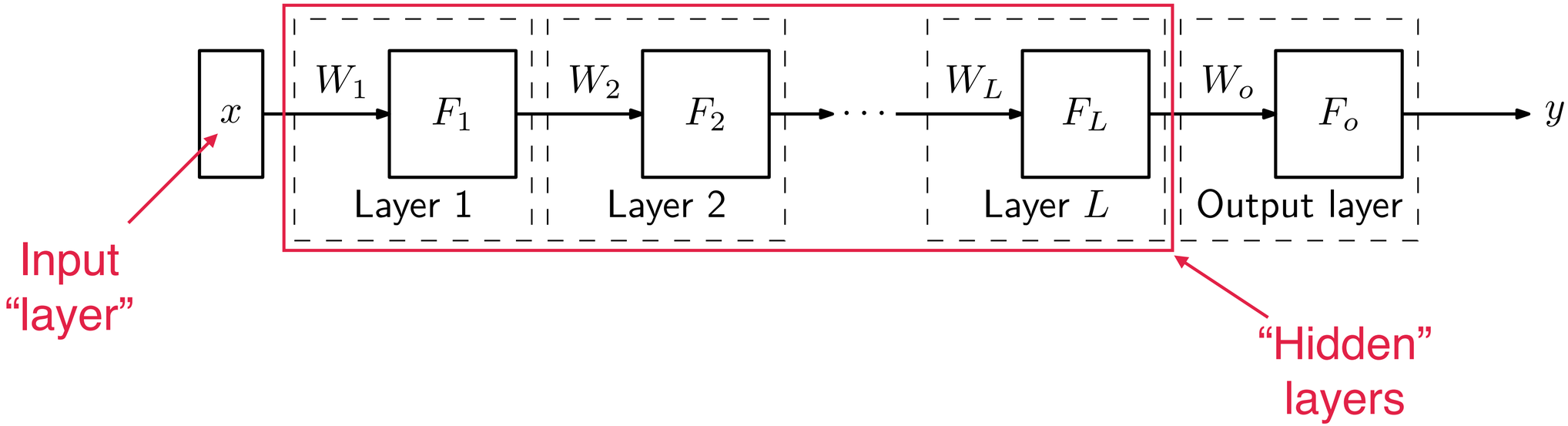
What does a neural network look like?

Feedforward network

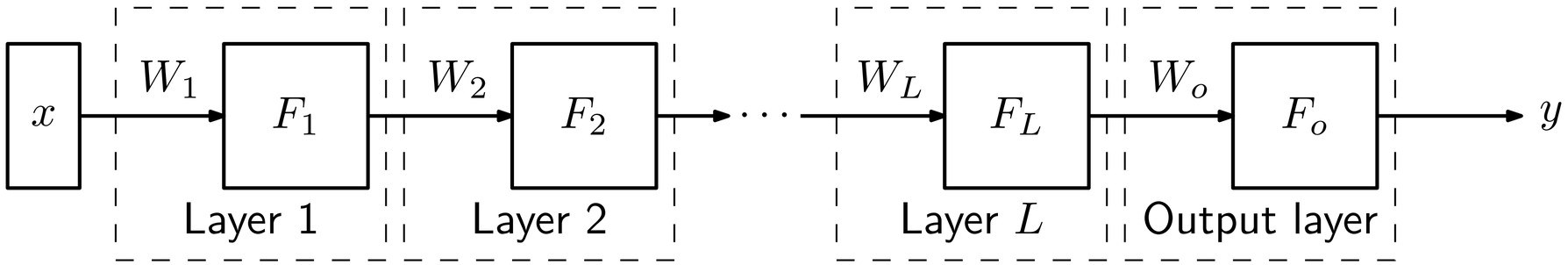


What does a neural network look like?

Feedforward network



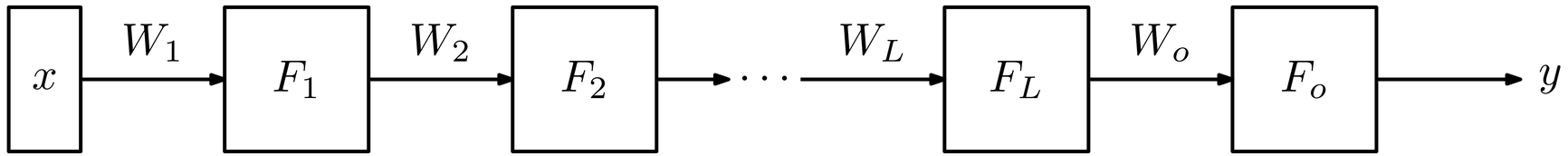
What does a neural network compute?



- If the network has *L* hidden layers, and h_i is the output of the *i*th hidden layer,
  - $h_1 = F_1(W_1 x)$
  - $h_2 = F_2(W_2 h_1) = F_2(W_2 F_1(W_1 x))$
  - …
  - $y = F_o(W_o h_L) = F_o(W_o F_L(W_L F_{L-1}(\ldots)))$

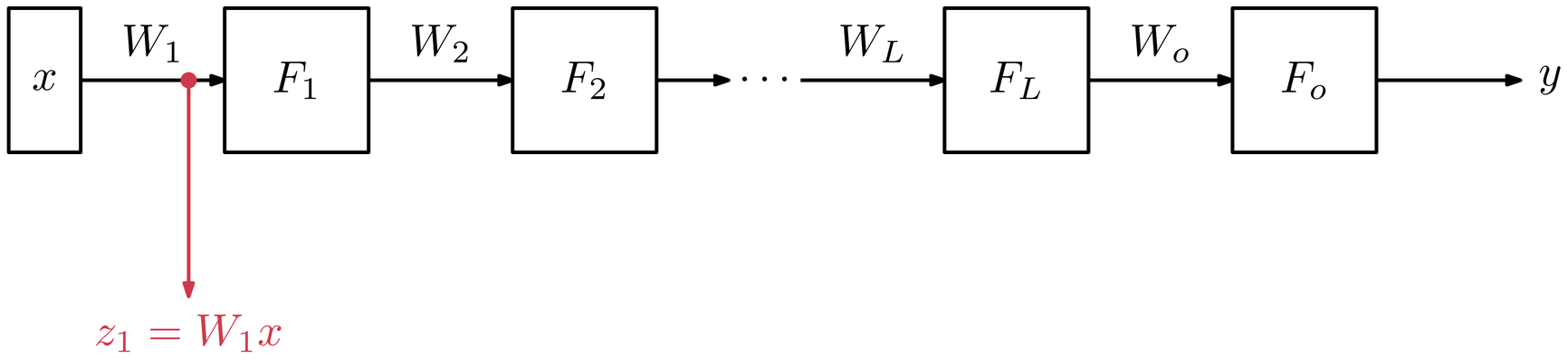
What does a neural network compute?

- The output *y* is (potentially) a complicated function of the input *x*
- Can be computed by propagating information forward through the network



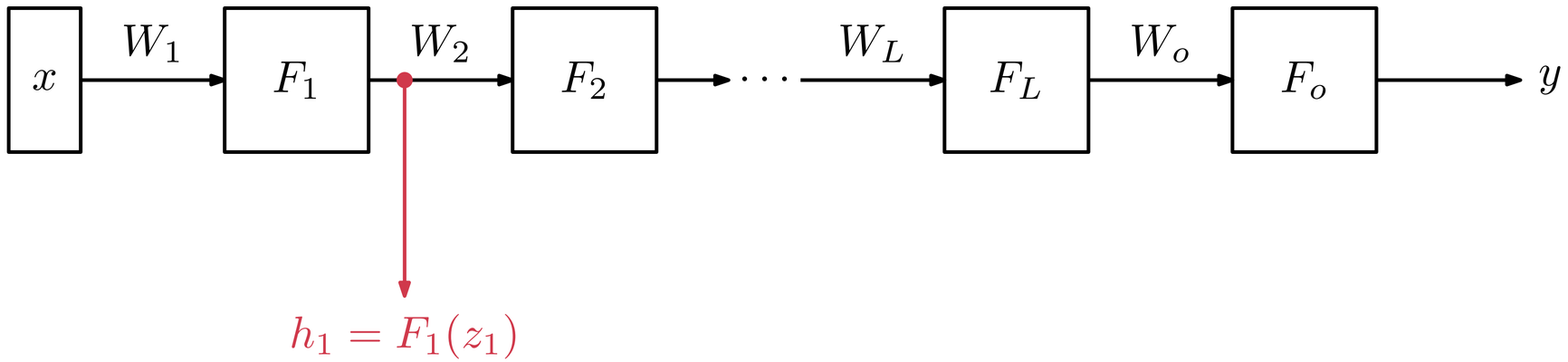
What does a neural network compute?

- The output *y* is (potentially) a complicated function of the input *x*
- Can be computed by propagating information forward through the network



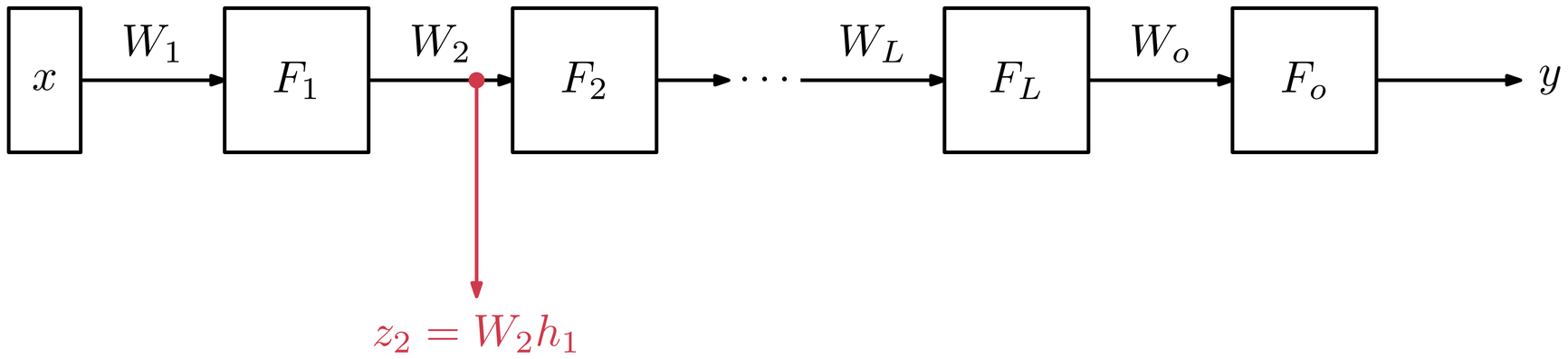
What does a neural network compute?

- The output *y* is (potentially) a complicated function of the input *x*
- Can be computed by propagating information forward through the network



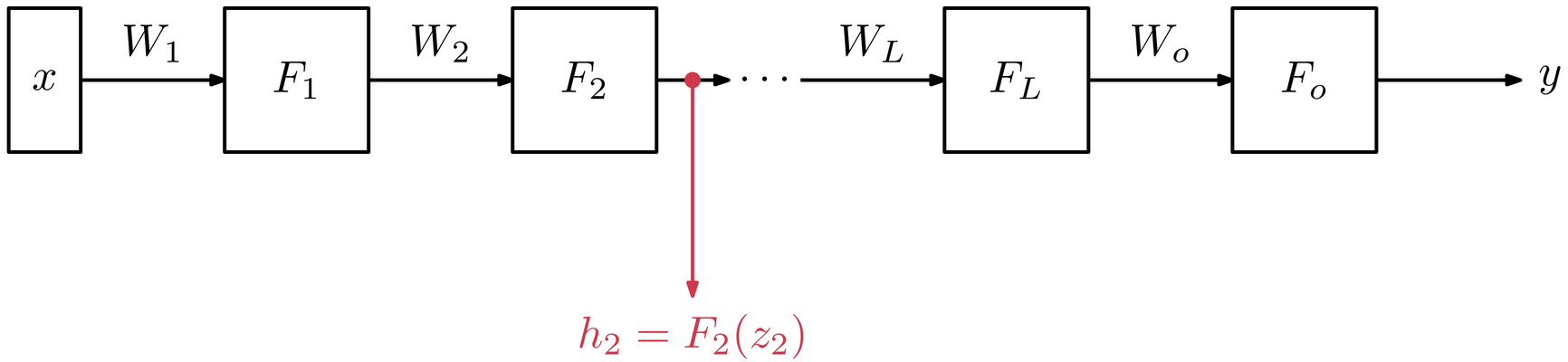
What does a neural network compute?

- The output *y* is (potentially) a complicated function of the input *x*
- Can be computed by propagating information forward through the network



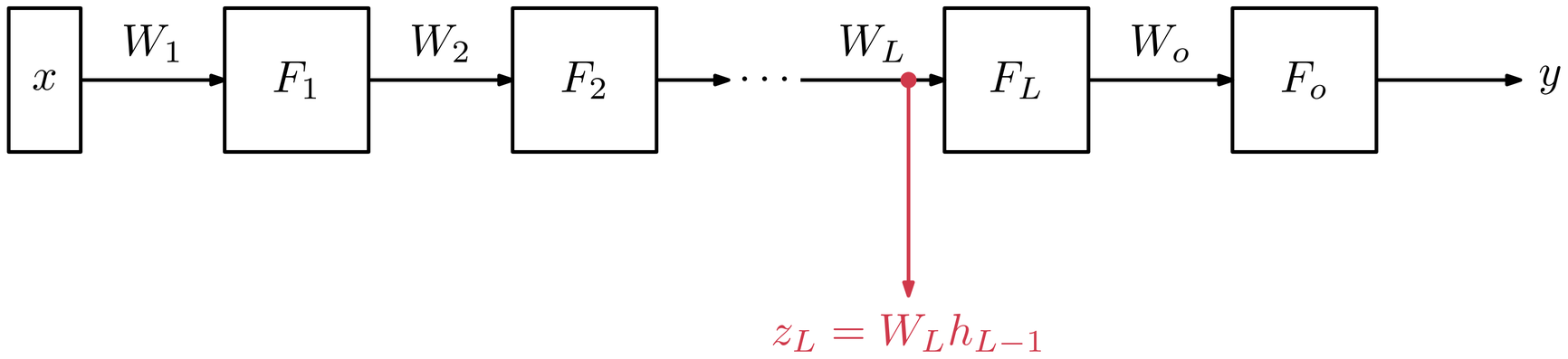
What does a neural network compute?

- The output *y* is (potentially) a complicated function of the input *x*
- Can be computed by propagating information forward through the network



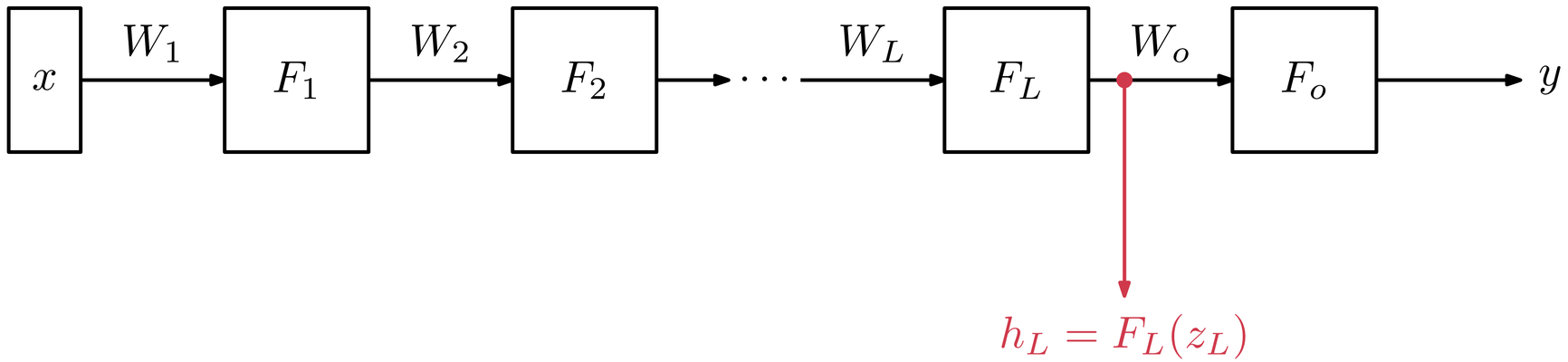
What does a neural network compute?

- The output *y* is (potentially) a complicated function of the input *x*
- Can be computed by propagating information forward through the network



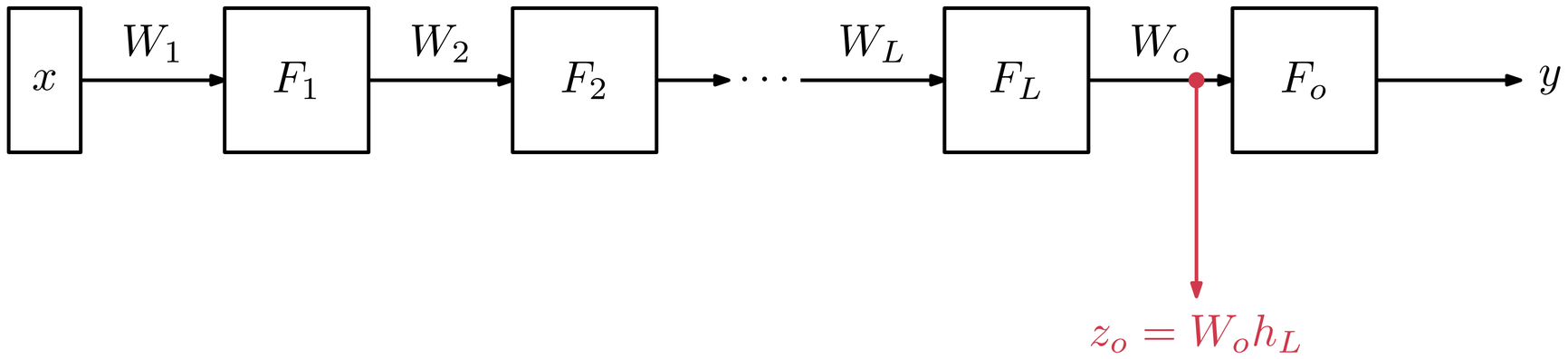
What does a neural network compute?

- The output *y* is (potentially) a complicated function of the input *x*
- Can be computed by propagating information forward through the network



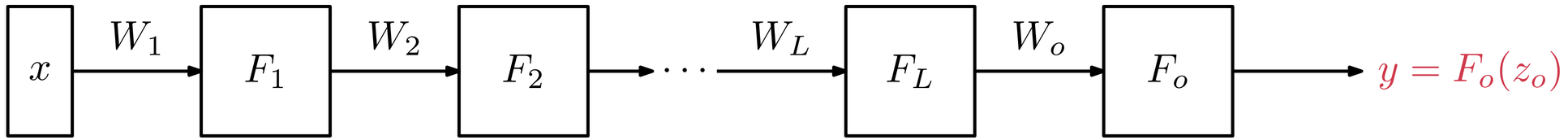
What does a neural network compute?

- The output *y* is (potentially) a complicated function of the input *x*
- Can be computed by propagating information forward through the network



What does a neural network compute?

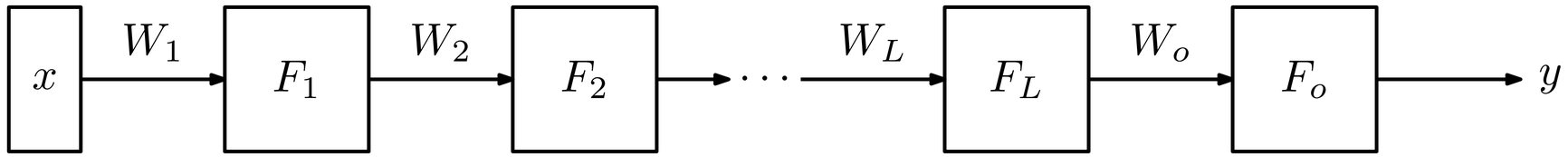
- The output *y* is (potentially) a complicated function of the input *x*
- Can be computed by propagating information forward through the network



Designing neural networks

… the alchemy of the modern days

Designing neural networks

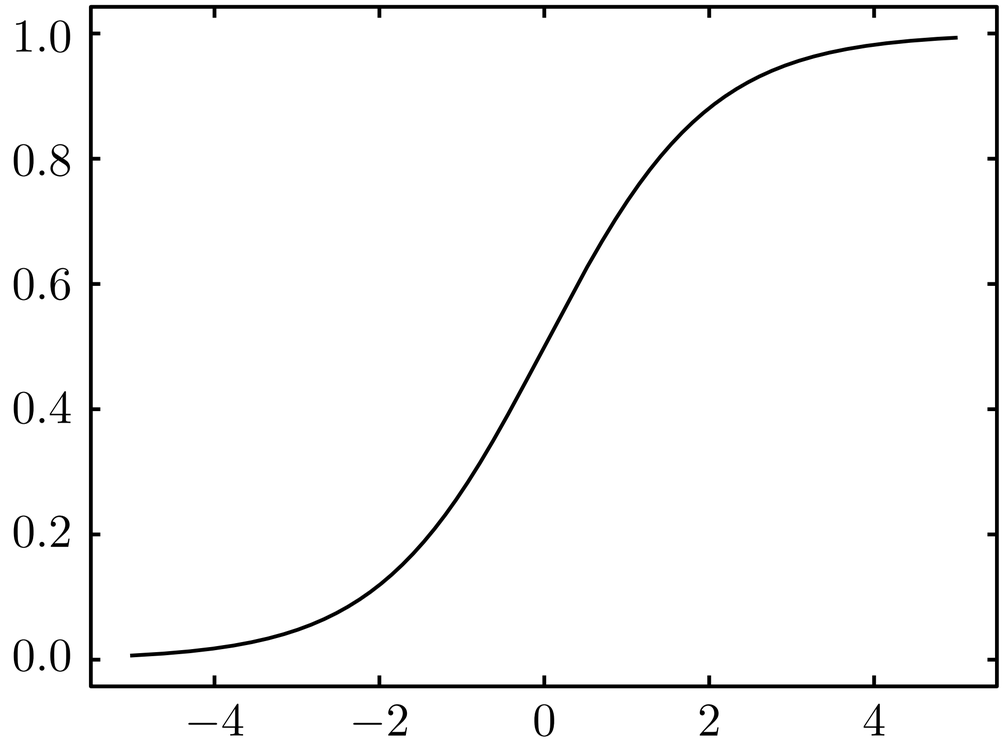


- Designing a neural network involves…
  - Selecting the number of layers, L
  - Selecting the number of units in each layer (i.e., the dimensions of W_1, …, W_L)
  - The activation functions, F_1, …, F_L
  - The output activation, F_o

Common activation functions

- **Logistic function:**

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

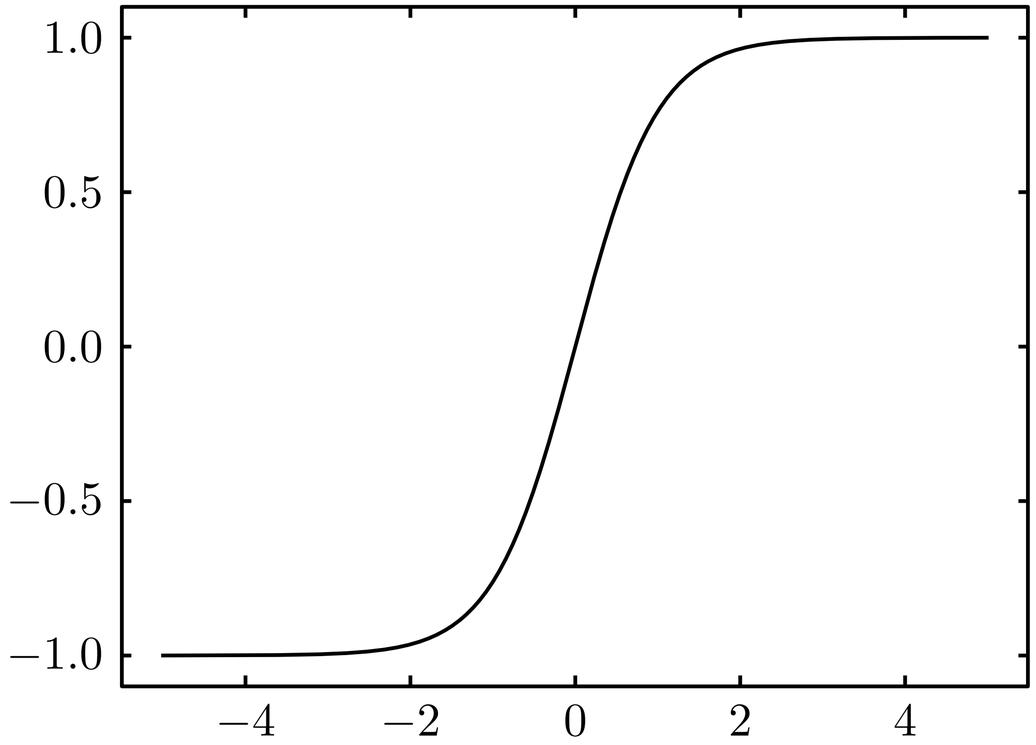


Common activation functions

- **Logistic function:**
  - Smooth version of threshold function
  - Squashes the input to (0, 1)
  - Can be interpreted as a probability
  - Used extensively in early days of neural networks

Common activation functions

- **Hyperbolic tangent:** $$\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$$



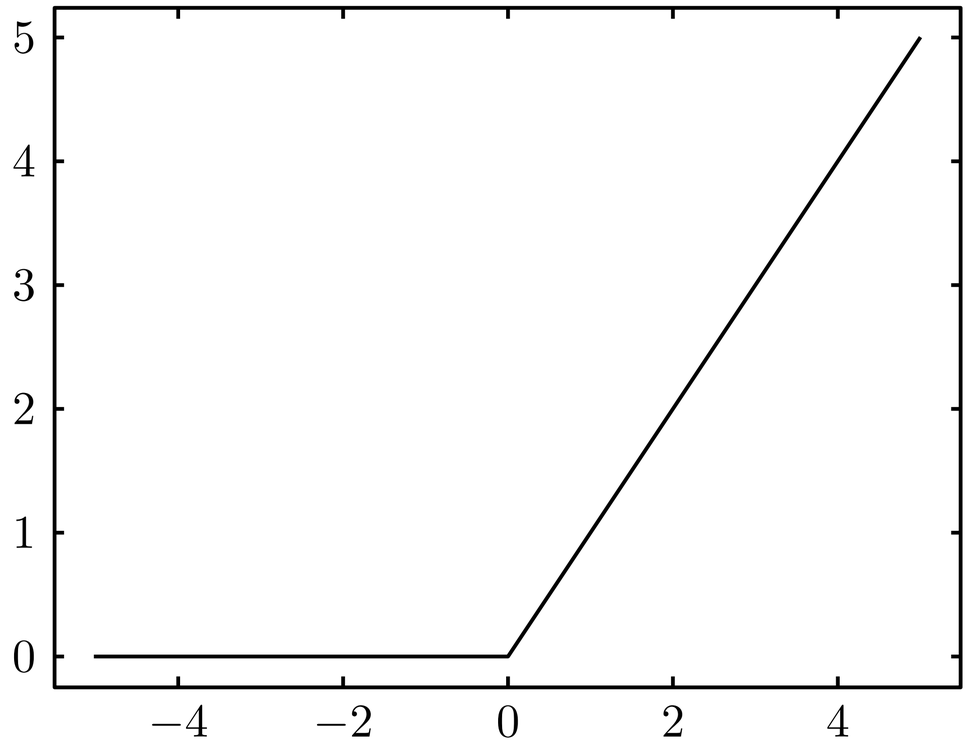
Common activation functions

- **Hyperbolic tangent:**
  - Another smooth version of threshold function
  - Squashes the input to (–1, 1)
  - Closely related with sigmoid
  - Also used extensively in early days of neural networks

$$\sigma(z) = \frac{1 + \tanh(z)}{2}$$

Common activation functions

- **Rectified linear unit:** $\mathrm{ReLU}(z) = \max\{0, z\}$



Common activation functions

- **Rectified linear unit:**
  - No positive saturation
  - Induces layers with sparse activity
  - Less prone to vanishing gradients
  - Used extensively in current practice

Common output functions

- **Logistic function:**

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

  - Used in binary classification problems
  - Output corresponds to probability of one of the classes

Common output functions

- **Softmax:**

$$[\text{softmax}(z)]_k = \frac{\exp(z_k)}{\sum_{k'=1}^{K} \exp(z_{k'})}$$

  - Multiclass “extension” of logistic function
  - Used in multiclass classification problems
  - Each output corresponds to the probability of each class

Common output functions

- **Linear:**

$$\text{linear}(z) = z$$

  - Commonly used in regression problems

- **Exponential:**

$$\exp(z) = e^z$$

  - Commonly used to learn positive quantities (e.g., variances)

Training neural networks

Training neural networks

- Minimize the empirical risk/regularized risk

$$\mathcal{L}(W) = \frac{1}{N} \sum_{n=1}^{N} \underbrace{L(x_n, y_n; W_{1:L}) + \lambda R(W_{1:L})}_{\mathcal{L}_n(W)}$$

- Use (stochastic) gradient descent:

$$W_\ell^{(k+1)} \leftarrow W_\ell^{(k)} - \alpha \sum_{n \in \mathcal{B}} \nabla_{W_\ell} \mathcal{L}_n(W), \qquad \ell = 1, \ldots, L$$

Batch

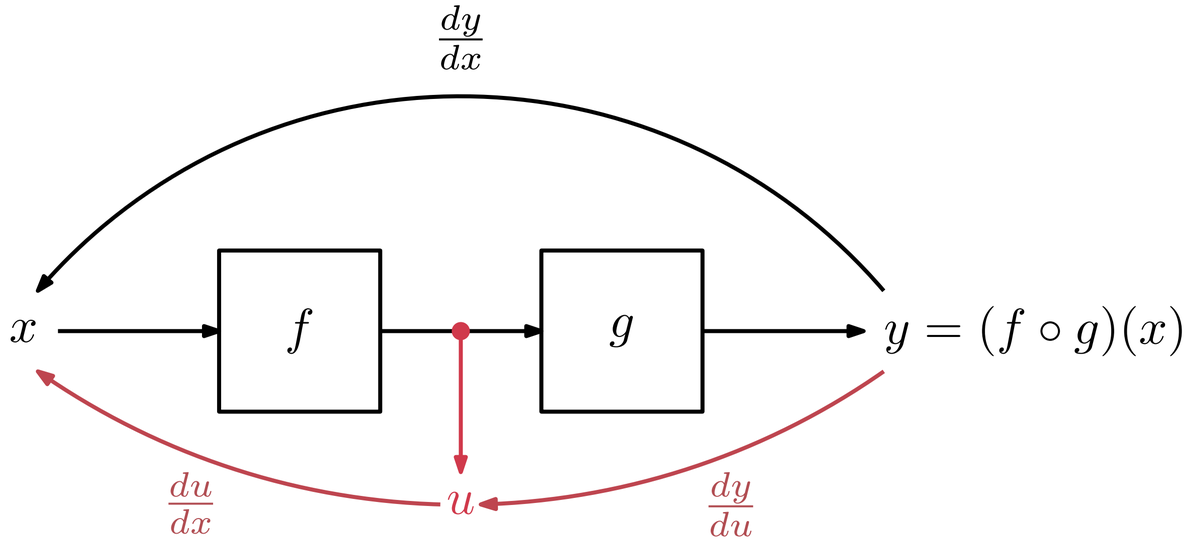
Computing the gradient

- Large networks have many parameters (often in the millions)
- How can we compute gradient?
  - Chain rule of derivatives:

$$\frac{d}{dx} f(g(x)) = \frac{d}{dg} f(g(x)) \frac{d}{dx} g(x)$$

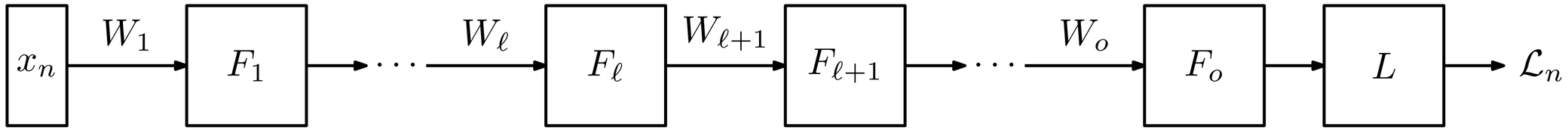
Computing the gradient

- Large networks have many parameters (often in the millions)
- How can we compute gradient?
  - Chain rule of derivatives:



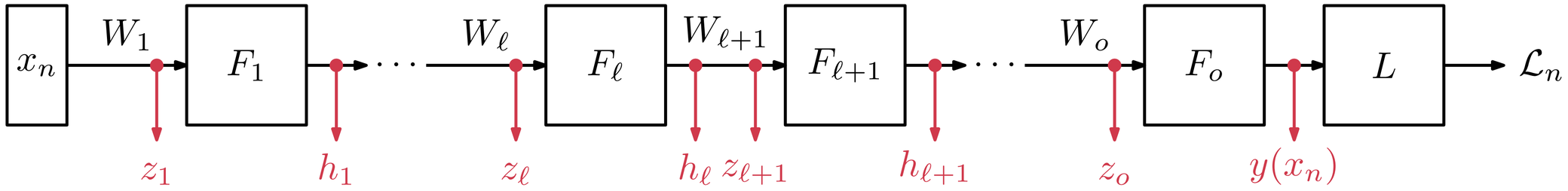
Computing the gradient

- In our neural network:



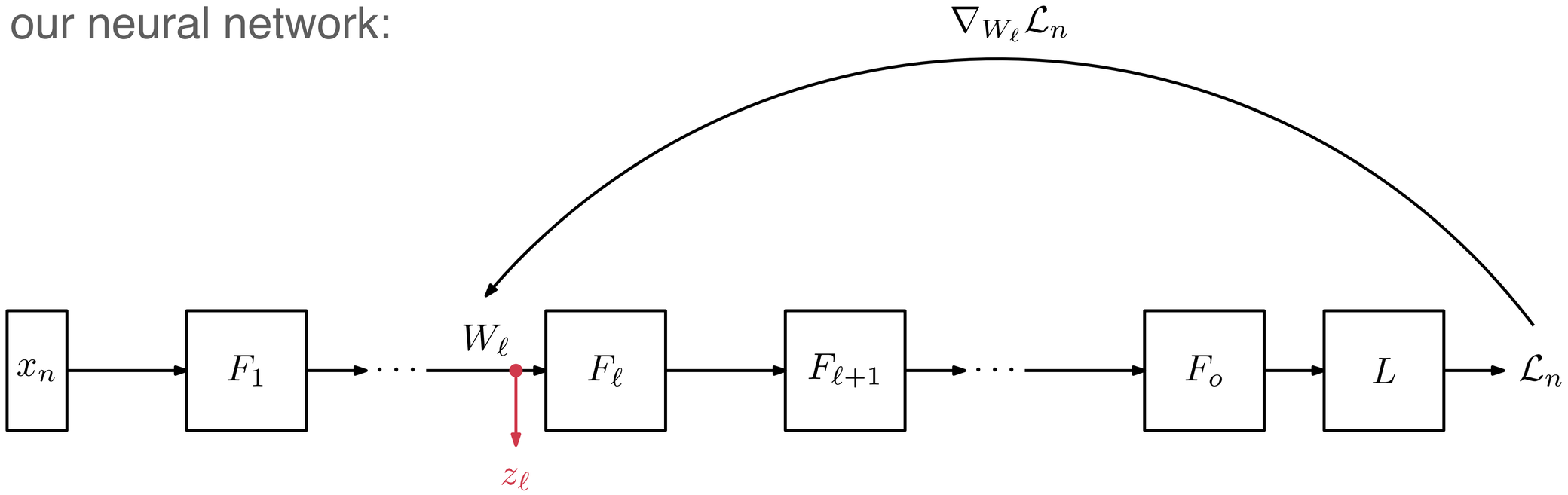
Computing the gradient

- In our neural network:



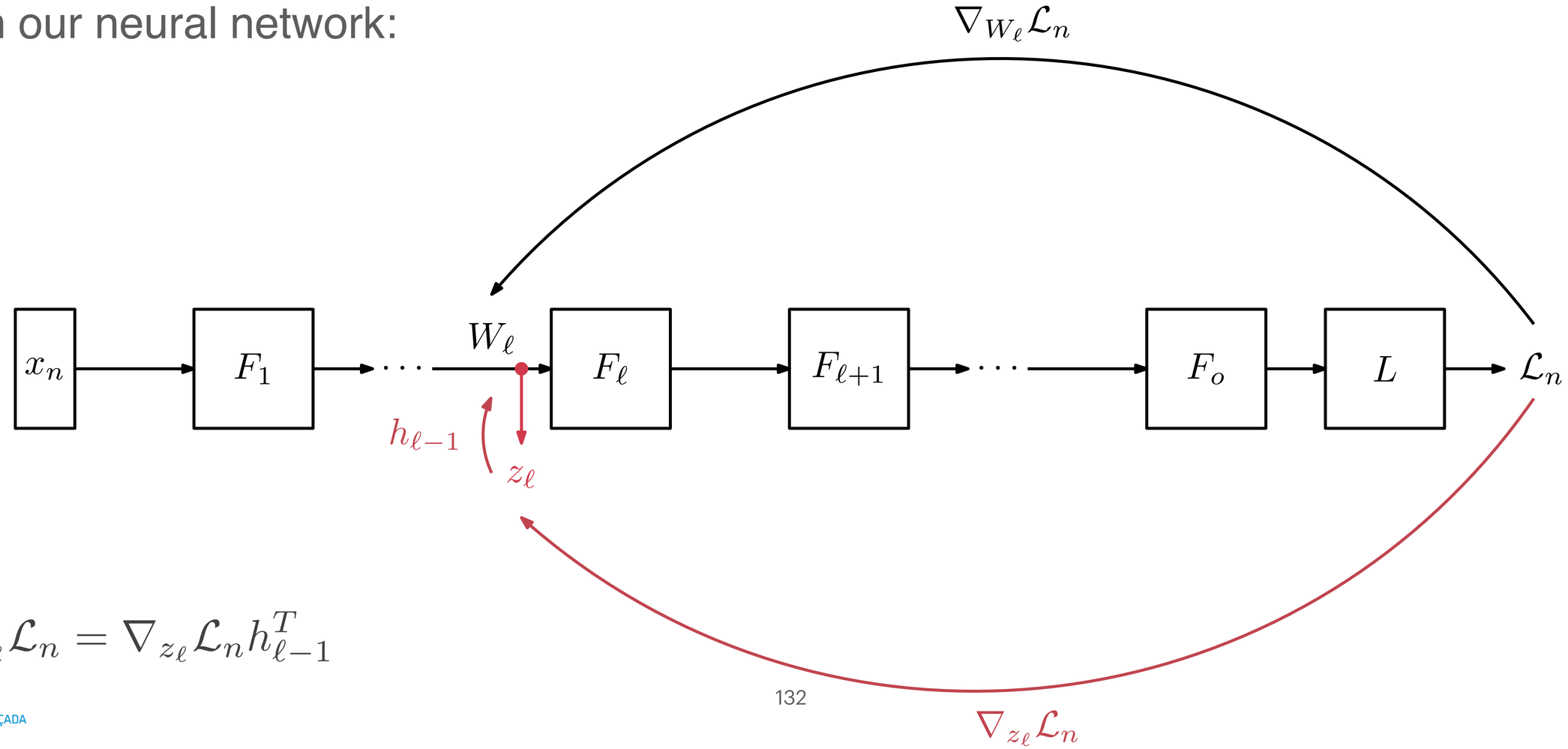
Computing the gradient

- In our neural network:



Computing the gradient

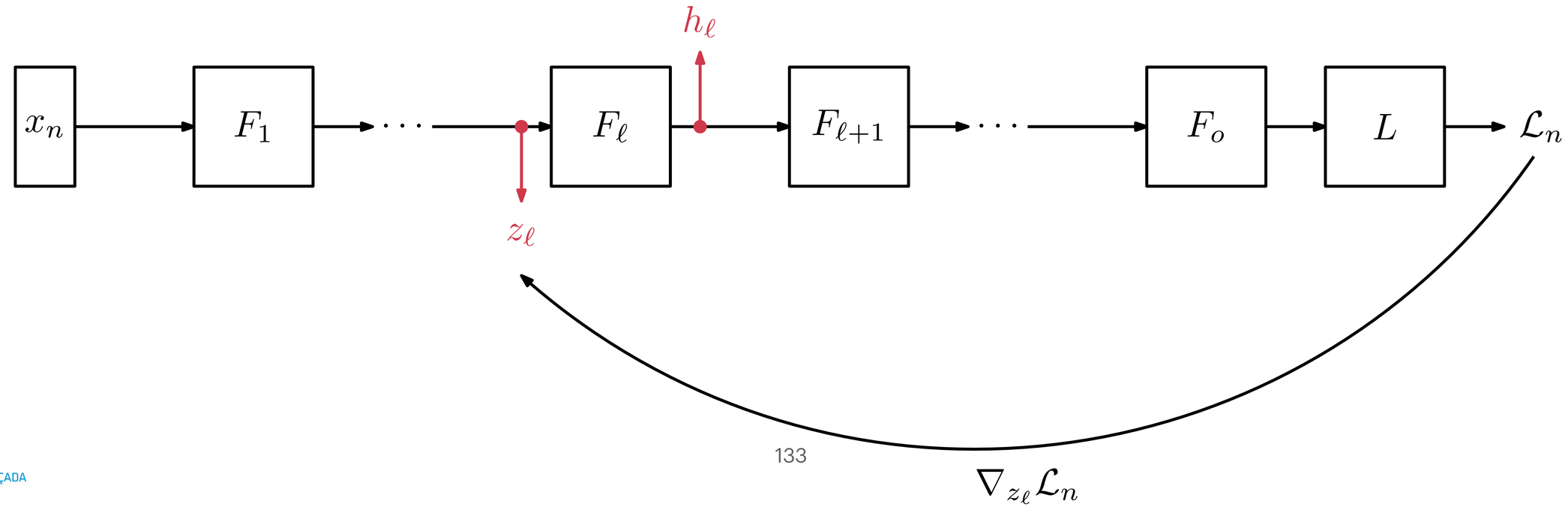
- In our neural network:



$$\nabla_{W_\ell} \mathcal{L}_n = \nabla_{z_\ell} \mathcal{L}_n h_{\ell-1}^T$$

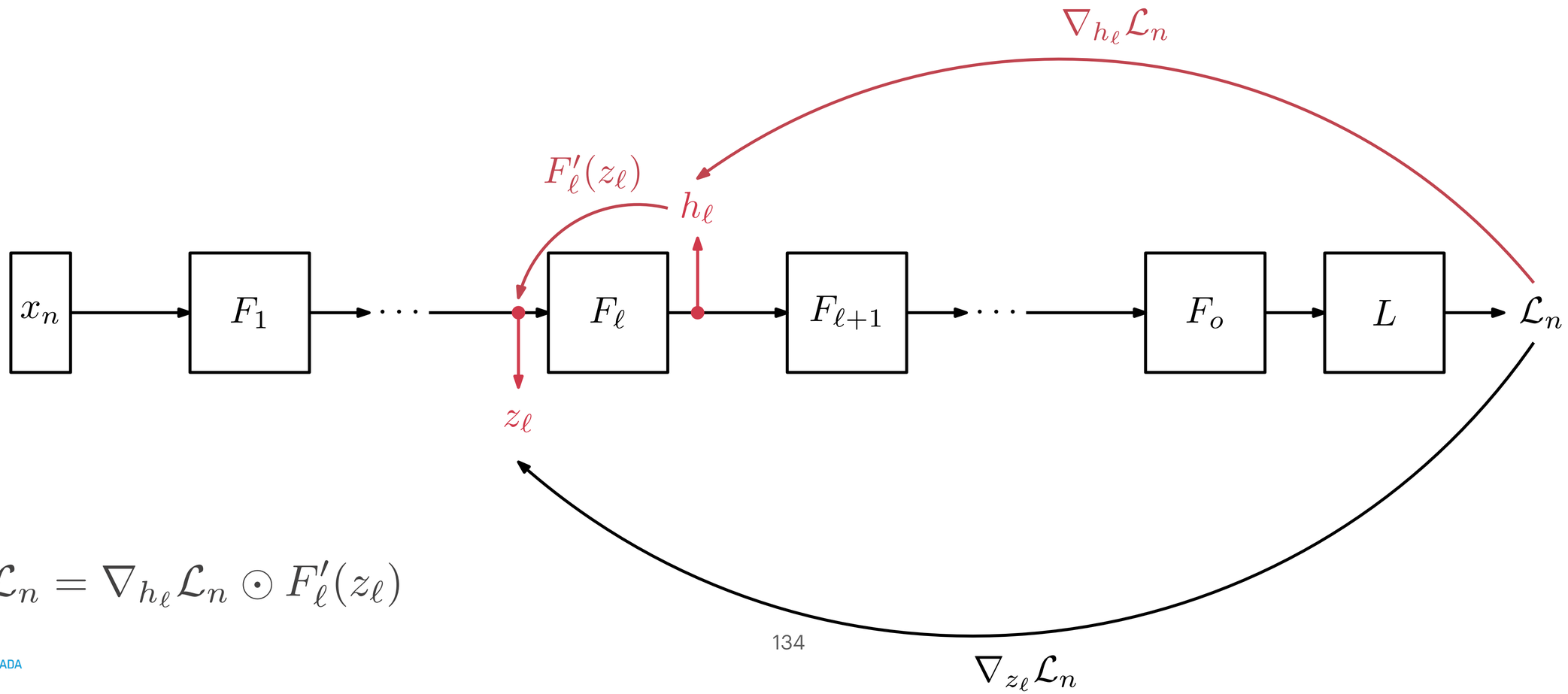
Computing the gradient

- In our neural network:



Computing the gradient

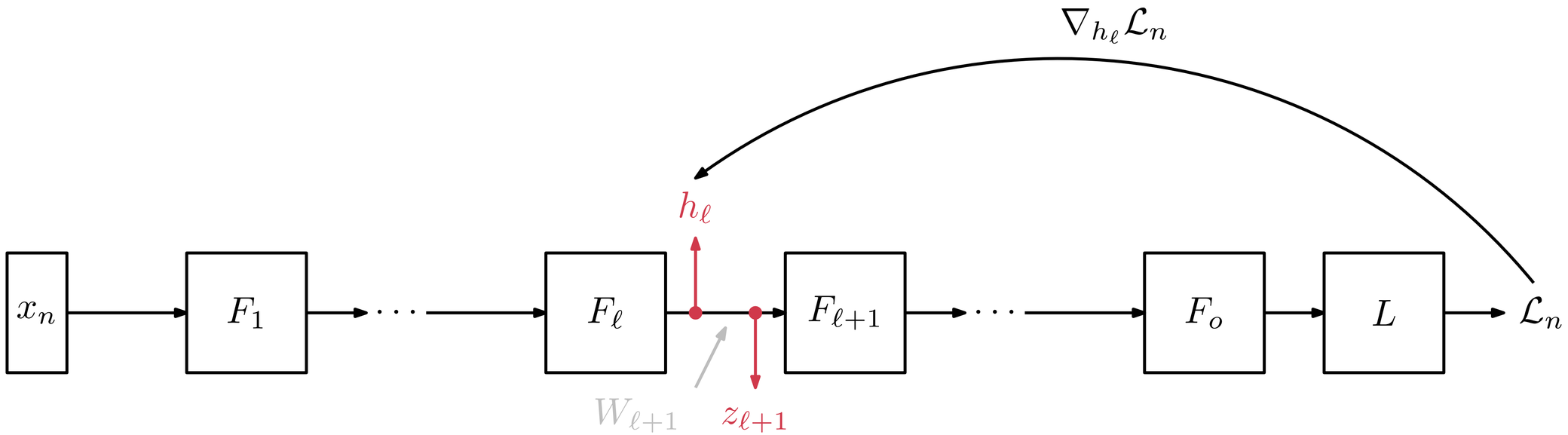
- In our neural network:



$$\nabla_{z_\ell} \mathcal{L}_n = \nabla_{h_\ell} \mathcal{L}_n \odot F'_\ell(z_\ell)$$

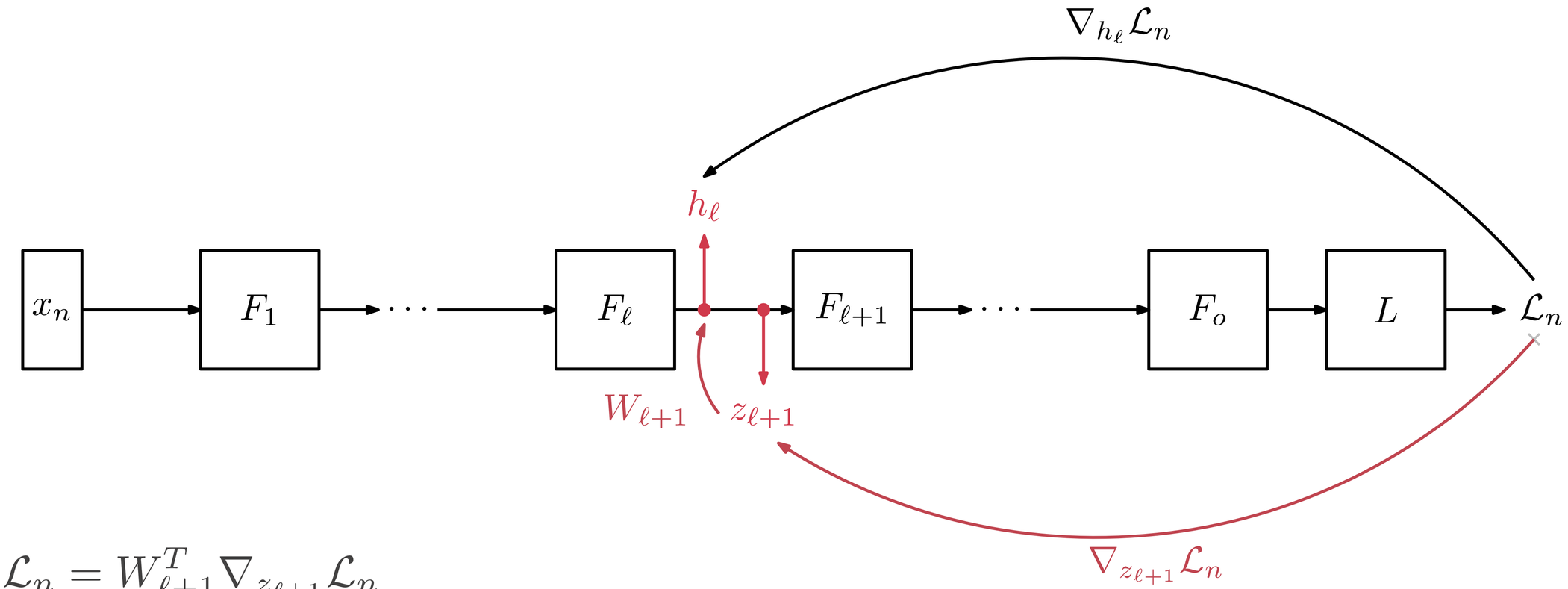
Computing the gradient

- In our neural network:



Computing the gradient

- In our neural network:



$$\nabla_{h_\ell} \mathcal{L}_n = W_{\ell+1}^T \nabla_{z_{\ell+1}} \mathcal{L}_n$$

Computing the gradient

- Putting everything together:

$$\nabla_{W_\ell} \mathcal{L}_n = \nabla_{z_\ell} \mathcal{L}_n h_{\ell-1}^T$$

$$\nabla_{z_\ell} \mathcal{L}_n = \nabla_{h_\ell} \mathcal{L}_n \odot F_\ell'(z_\ell)$$

$$\nabla_{h_\ell} \mathcal{L}_n = W_{\ell+1}^T \nabla_{z_{\ell+1}} \mathcal{L}_n$$

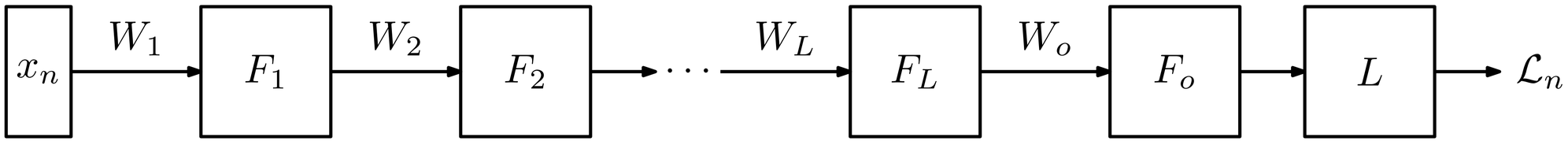
Computing the gradient

- Putting everything together:
  - $\nabla_{W_o} \mathcal{L}_n = \nabla_{z_o} \mathcal{L}_n h_L^T,$ with $\nabla_{z_o} \mathcal{L}_n = \text{diff}(y(x_n), y_n)$
  - $\nabla_{W_L} \mathcal{L}_n = \nabla_{z_L} \mathcal{L}_n h_{L-1}^T,$ with $\nabla_{z_L} \mathcal{L}_n = \nabla_{h_L} \mathcal{L}_n \odot F_L'(z_L)$ and $\nabla_{h_L} \mathcal{L}_n = W_o^T \nabla_{z_o} \mathcal{L}_n$
  - ...
  - $\nabla_{W_1} \mathcal{L}_n = \nabla_{z_1} \mathcal{L}_n x,$ with $\nabla_{z_1} \mathcal{L}_n = \nabla_{h_1} \mathcal{L}_n \odot F_1'(z_1)$ and $\nabla_{h_1} \mathcal{L}_n = W_2^T \nabla_{z_2} \mathcal{L}_n$

Already computed

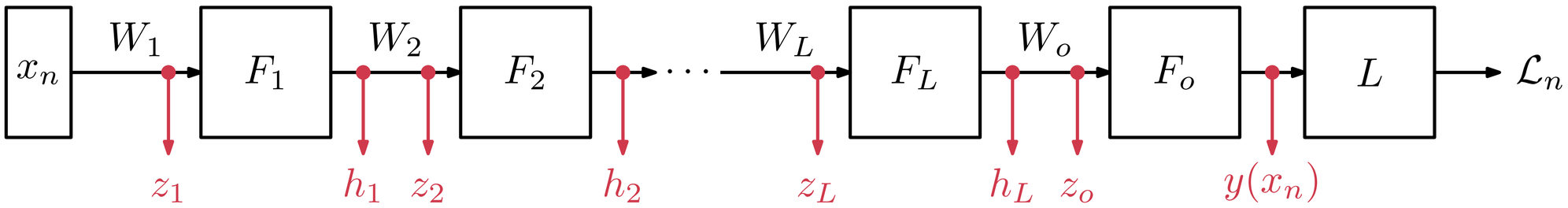
Computing the gradient

- Putting everything together:



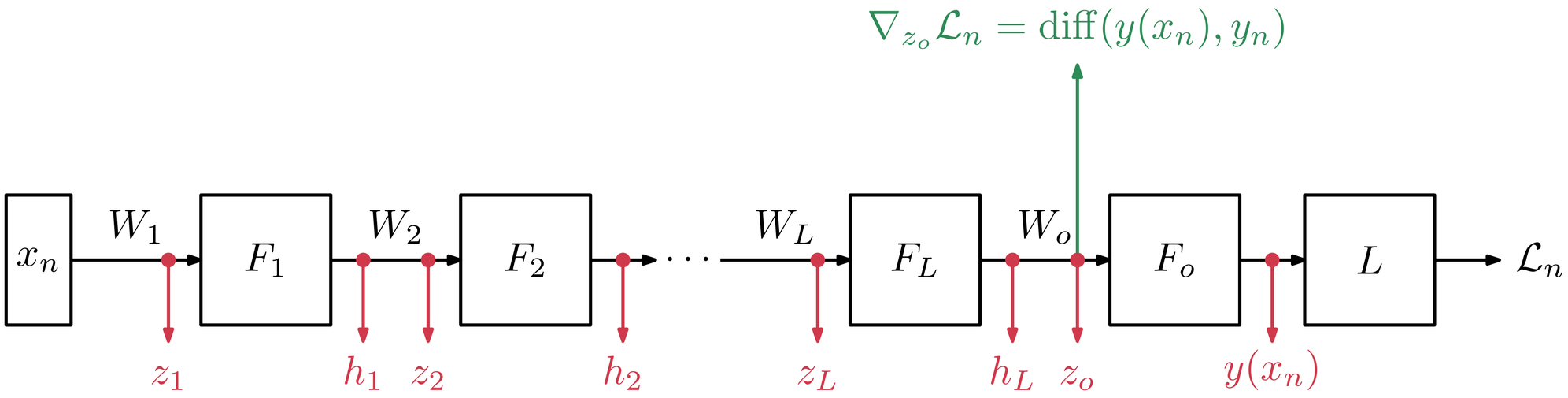
Computing the gradient

- Putting everything together:



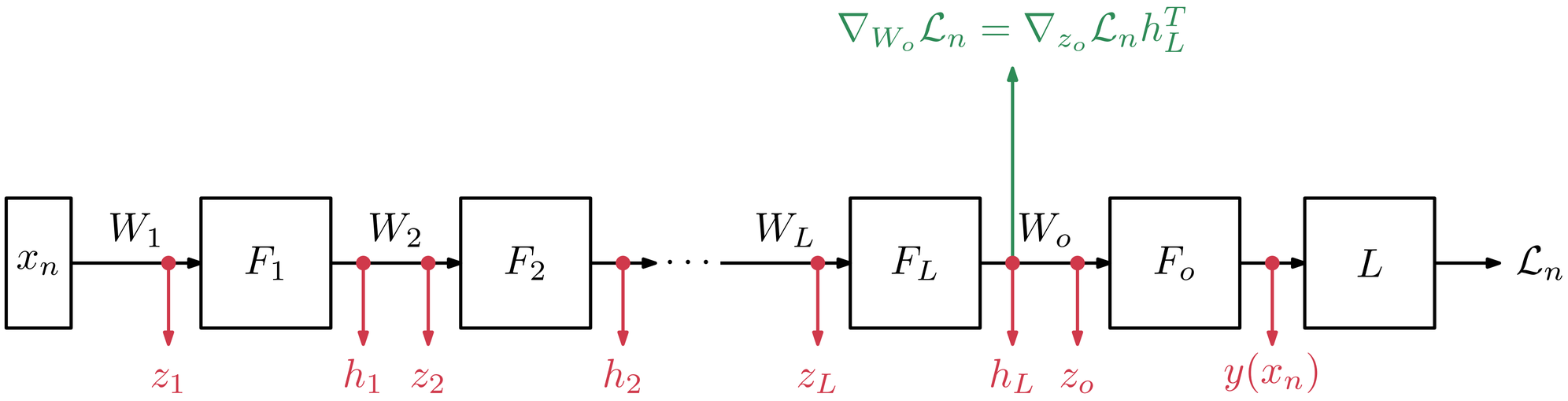
Computing the gradient

- Putting everything together:



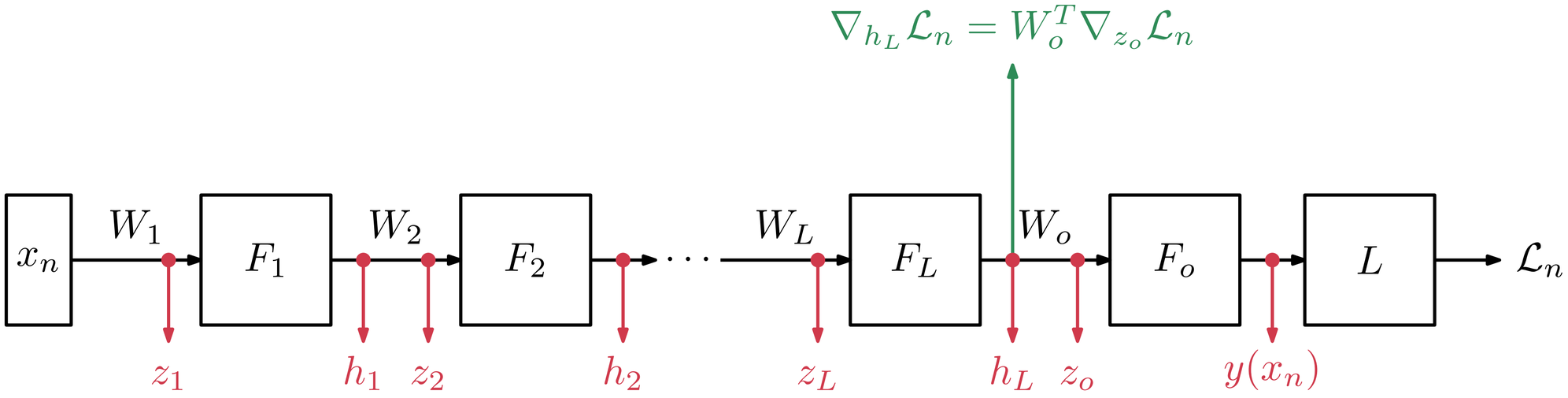
Computing the gradient

- Putting everything together:



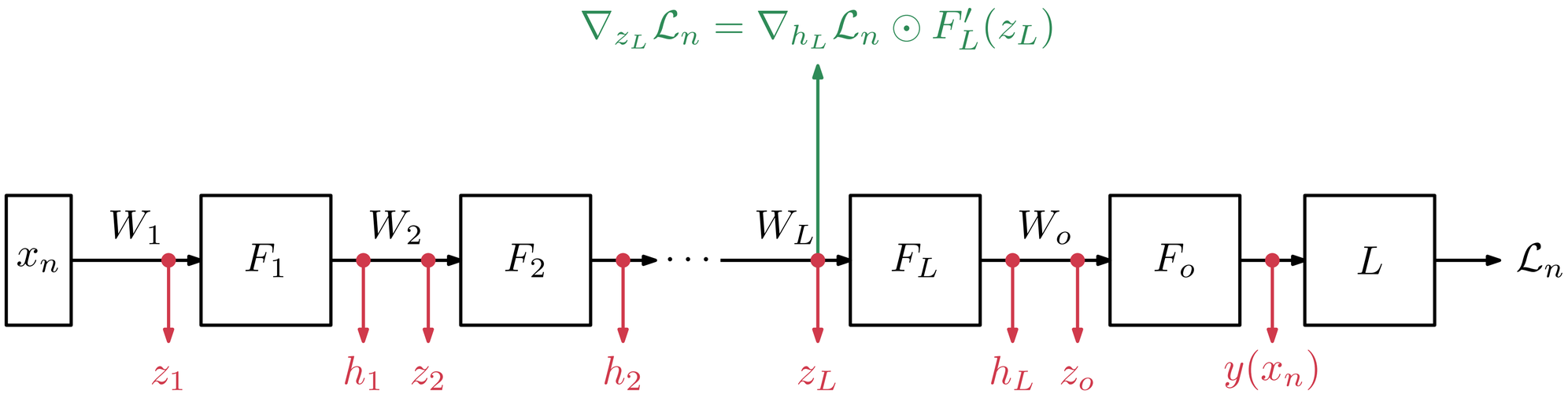
Computing the gradient

- Putting everything together:



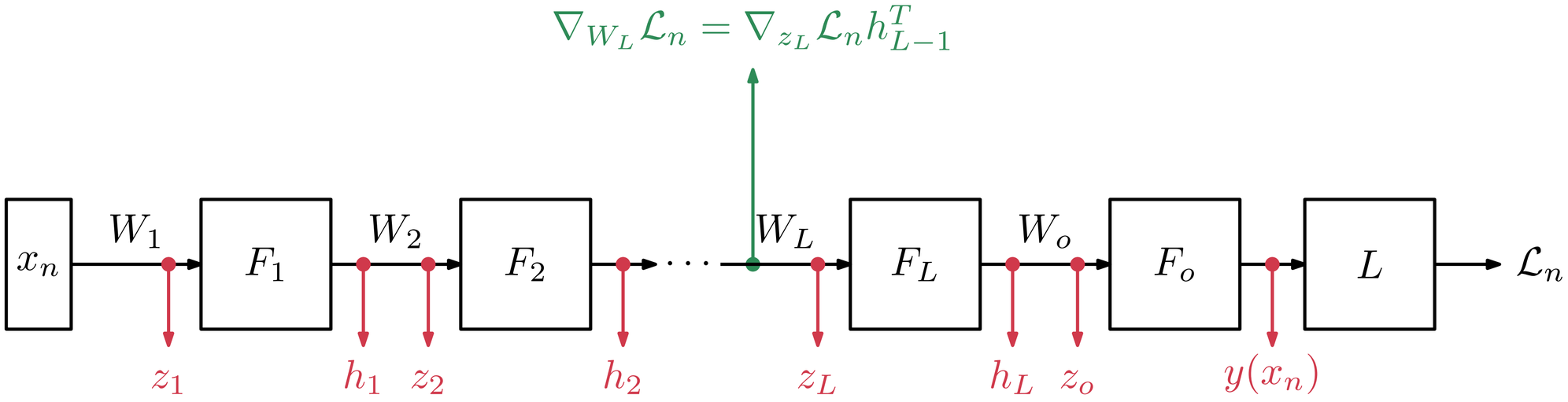
Computing the gradient

- Putting everything together:



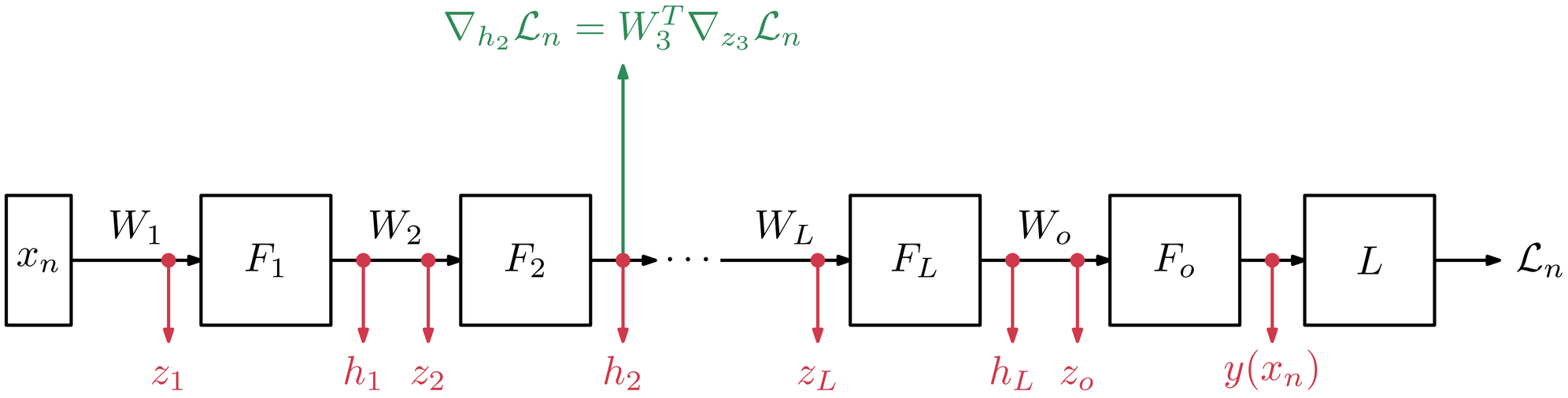
Computing the gradient

- Putting everything together:



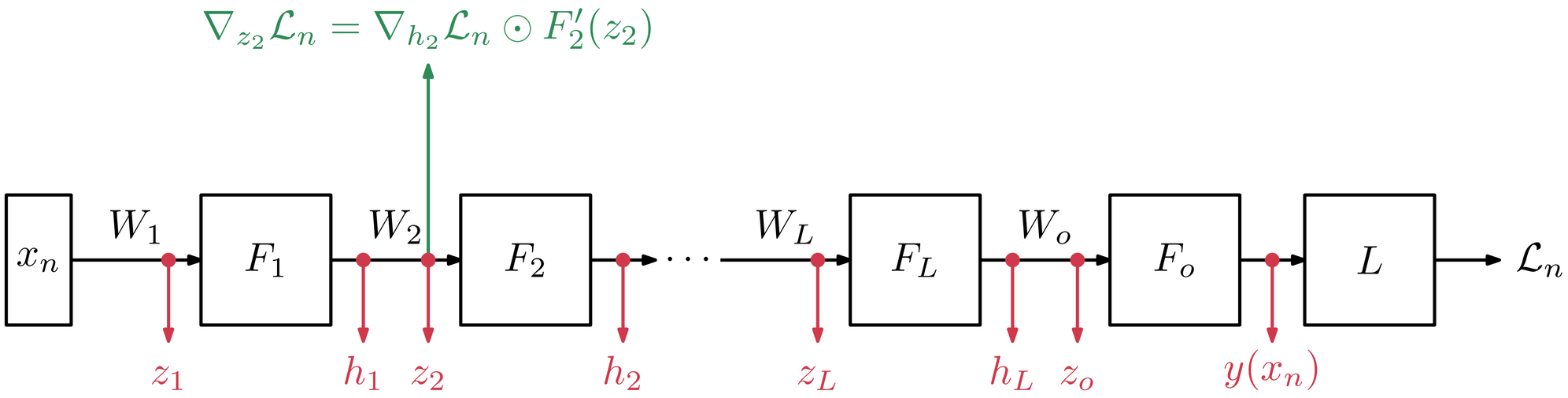
Computing the gradient

- Putting everything together:



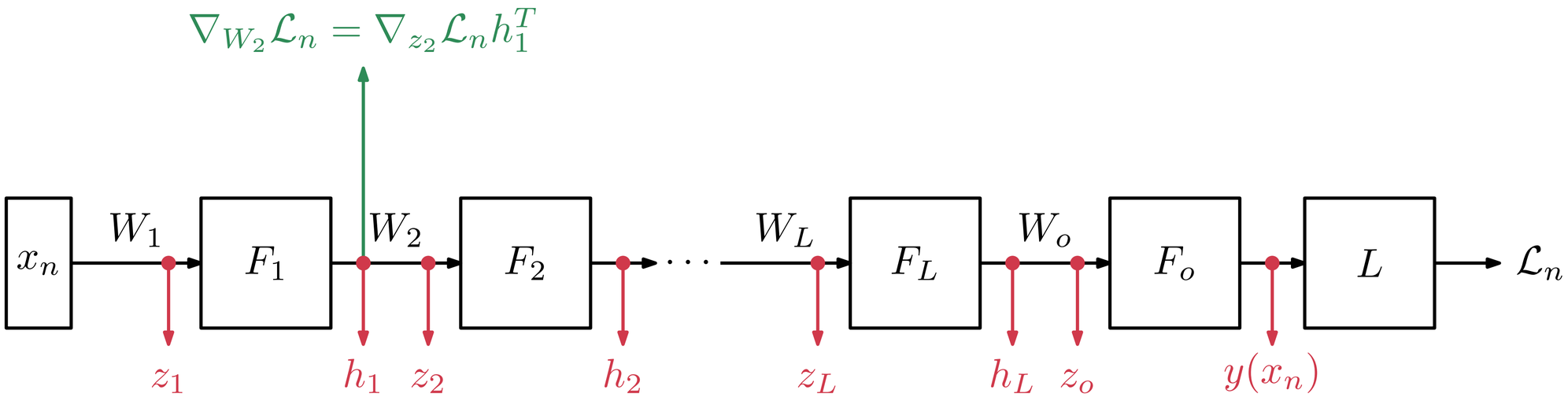
Computing the gradient

- Putting everything together:



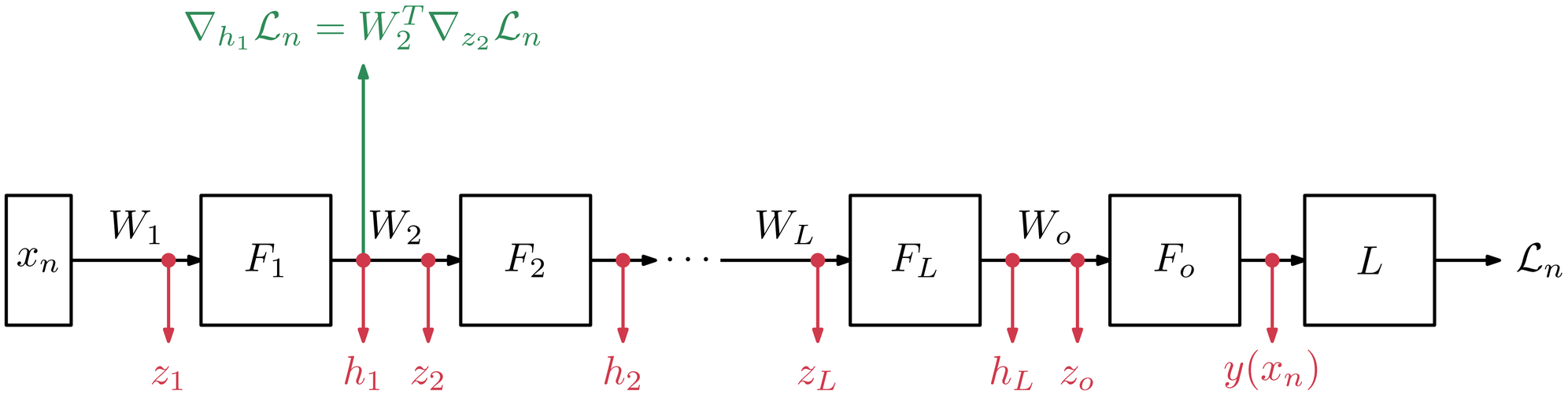
Computing the gradient

- Putting everything together:



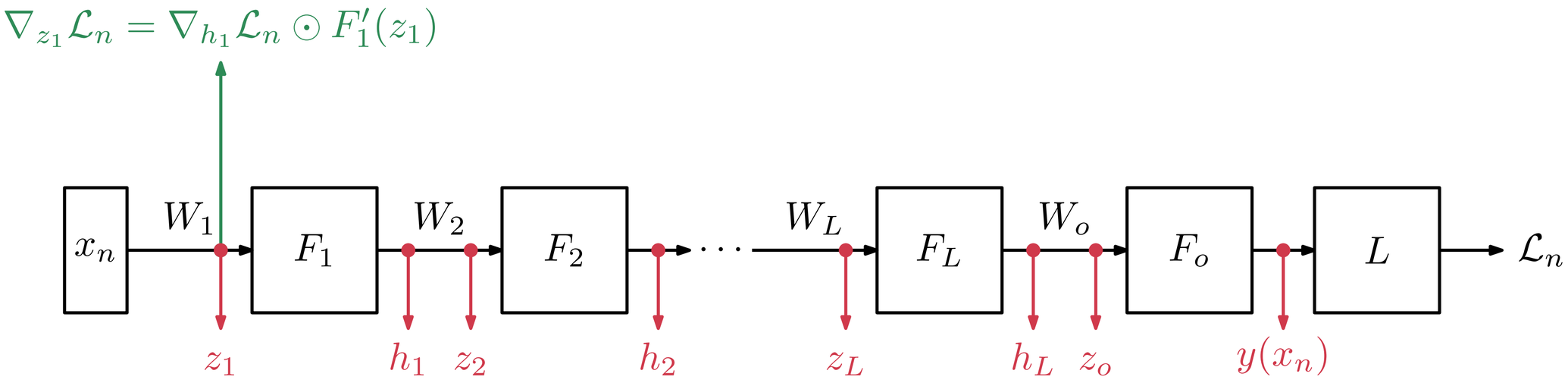
Computing the gradient

- Putting everything together:



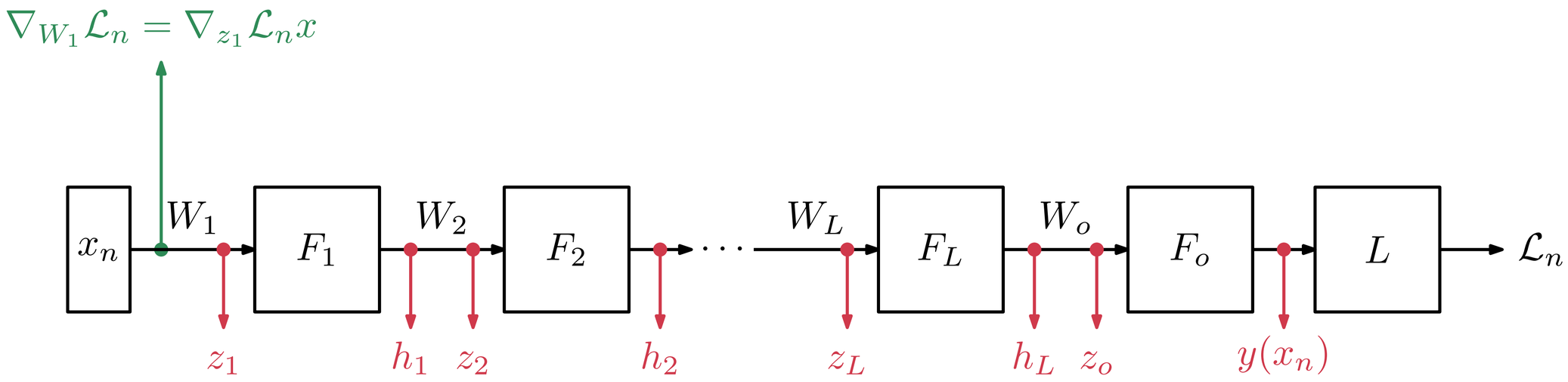
Computing the gradient

- Putting everything together:



Computing the gradient

- Putting everything together:



Computing the gradient

- The gradient is computed by propagating gradient information backward through the network
- This process is known as back-propagation
  - It is a general approach, used to compute gradient in computational graphs
  - It requires keeping track of intermediate quantities (e.g., the different z_i and h_i)
  - Auto-differentiation engines are designed to perform these computations efficiently

In summary…

Steps involved in deploying a deep learning system

- Figuring out the data to be used (Experience *E*)
- Figuring out the network architecture to be used (Task *T*)
- Figuring out the loss function to be used (and regularization) (Performance *P*)
- Train the network using gradient descent and back-propagation (using auto-diff)

More on regularization…

More on regularization…

- Recall:
  - In regularized risk minimization, we are willing to sacrifice accuracy in the training data to have a “simpler” model

$$\frac{1}{N}\sum_{n=1}^{N}\left(L(x_n, y_n; W_{1:L}) + \lambda R(W_{1:L})\right)$$

  - The regularization term penalizes models with more “complex” parameters

More on regularization…

- Examples of regularization:
  - L^2 regularization:

$$R(W) = \frac{1}{2}\|W\|_2^2$$

  - Penalizes models where the parameters are far from the origin

More on regularization…

- Examples of regularization:
  - L^1 regularization:

$$R(W) = \|W\|_1$$

  - Penalizes models with non-sparse weights

More on regularization…

- It is also possible to perform regularization by “interfering” with the training process, making it harder for the network to learn
- Examples:
  - Data augmentation — Create “fake” data from the data available
  - Adding noise — We add noise to inputs, or weights, or targets
  - Early stopping — Prevent the network from learning for “too long”

More on regularization…

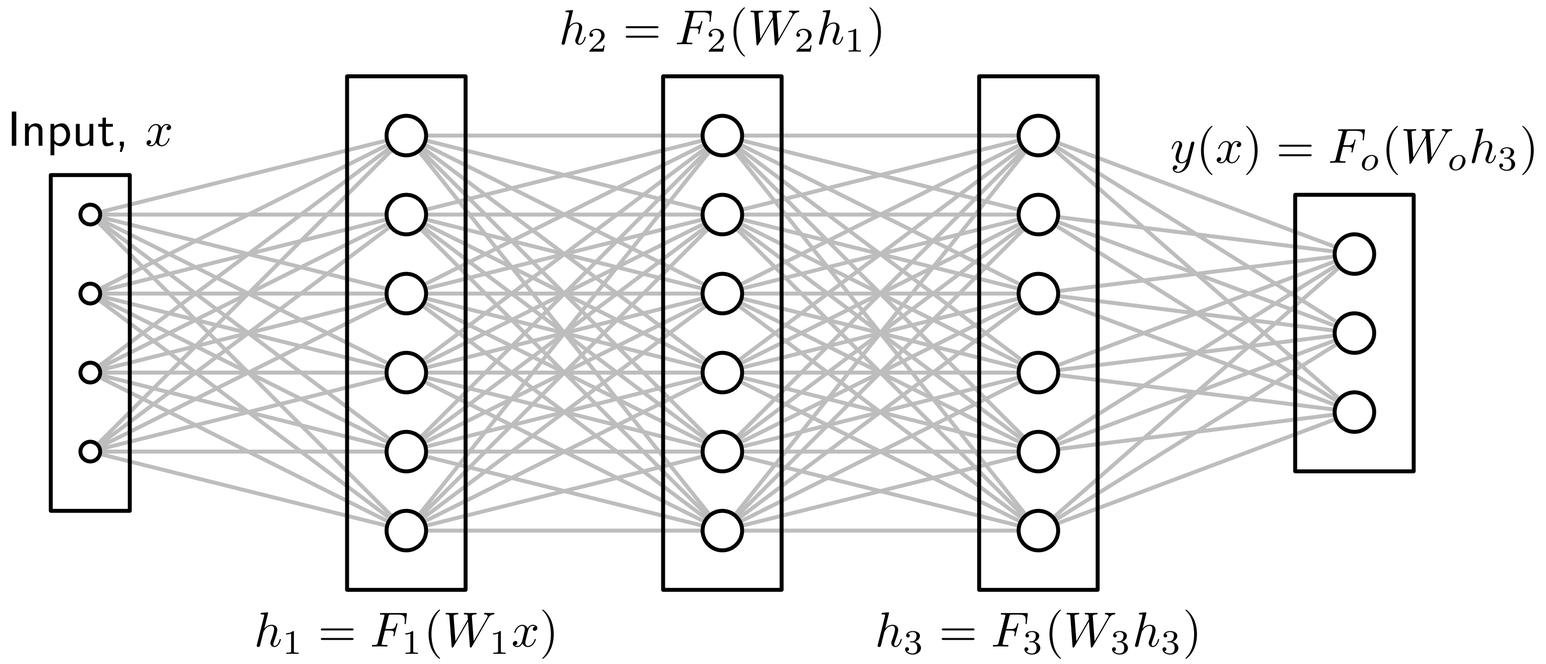
- More examples:
  - Parameter tying — Penalizing deviations from previous values (similar to norm penalty)
  - Parameter sharing — Forcing different units to have common weights (we'll see more of this when we discuss CNNs)
  - Dropout

Dropout

- Dropout is widely used in present day deep learning practice
- Dropout “turns off” random units during training, to make the network more robust to unit specialization
  - Each unit is “turned off” with probability *p* (dropout probability)
  - Similar to trainining multiple networks with shared weights

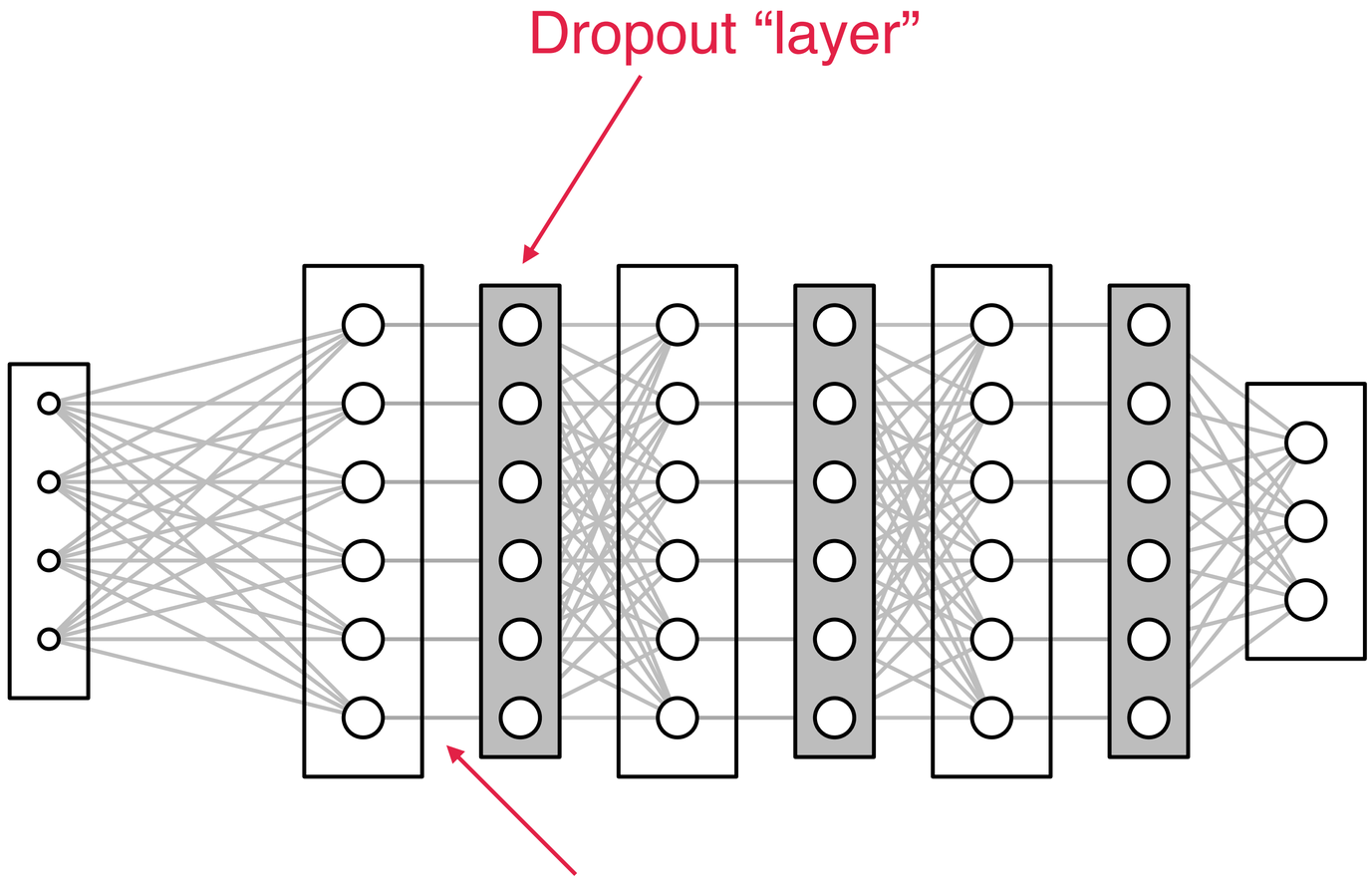
Dropout

- Example:



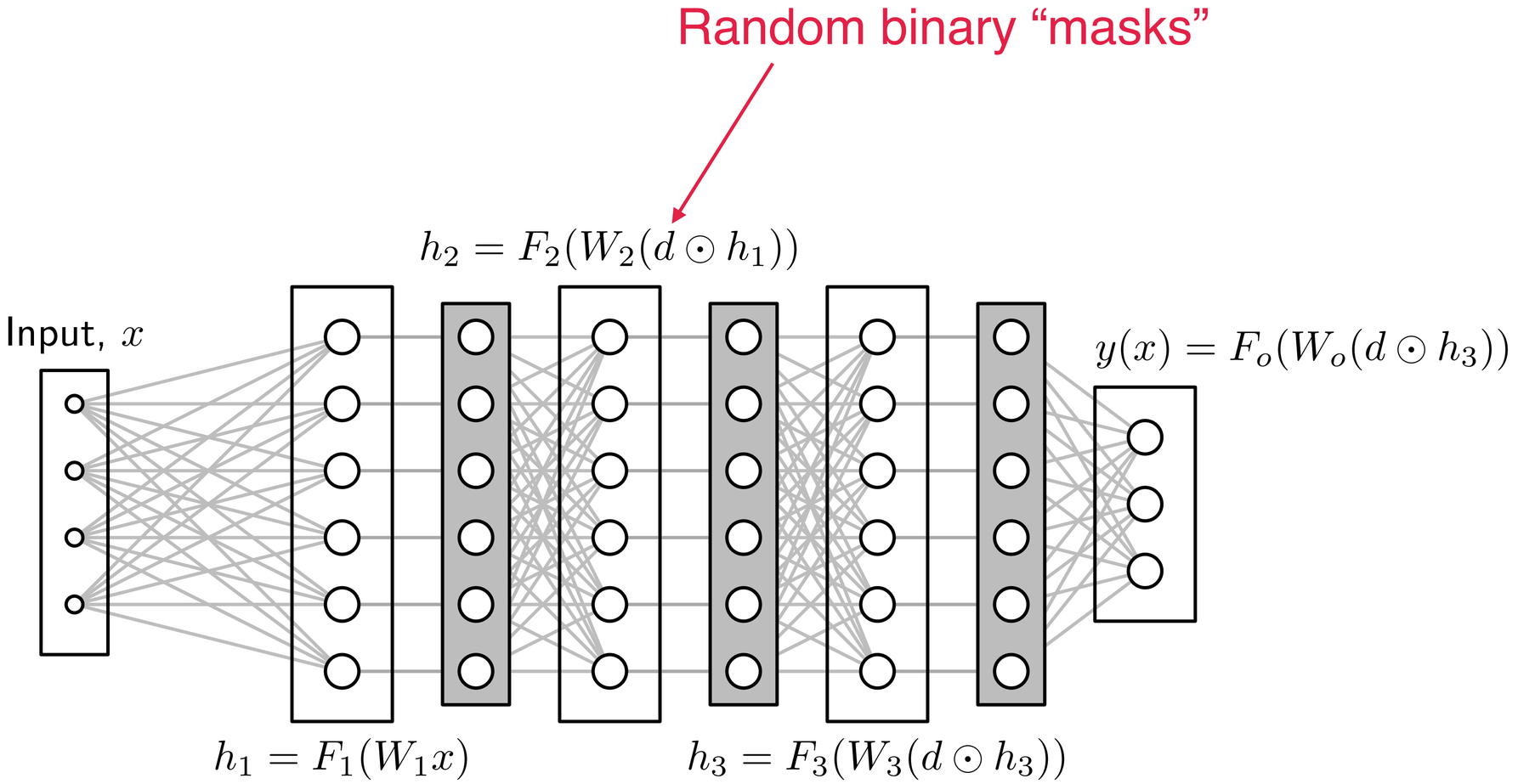
Dropout

- Example:



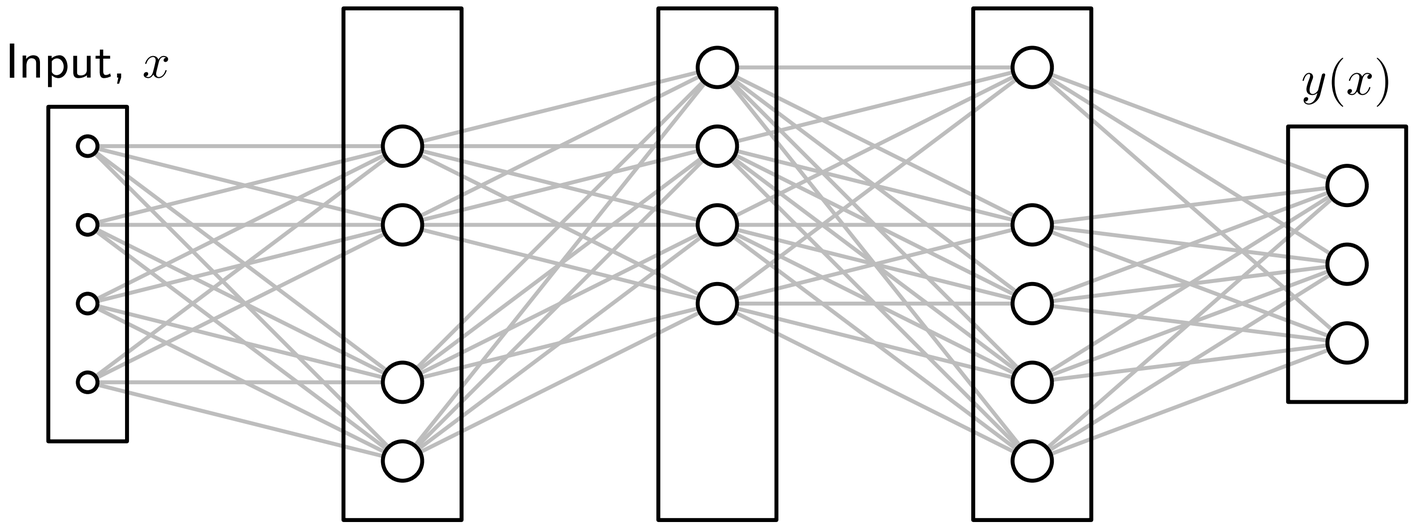
Dropout

- Example:



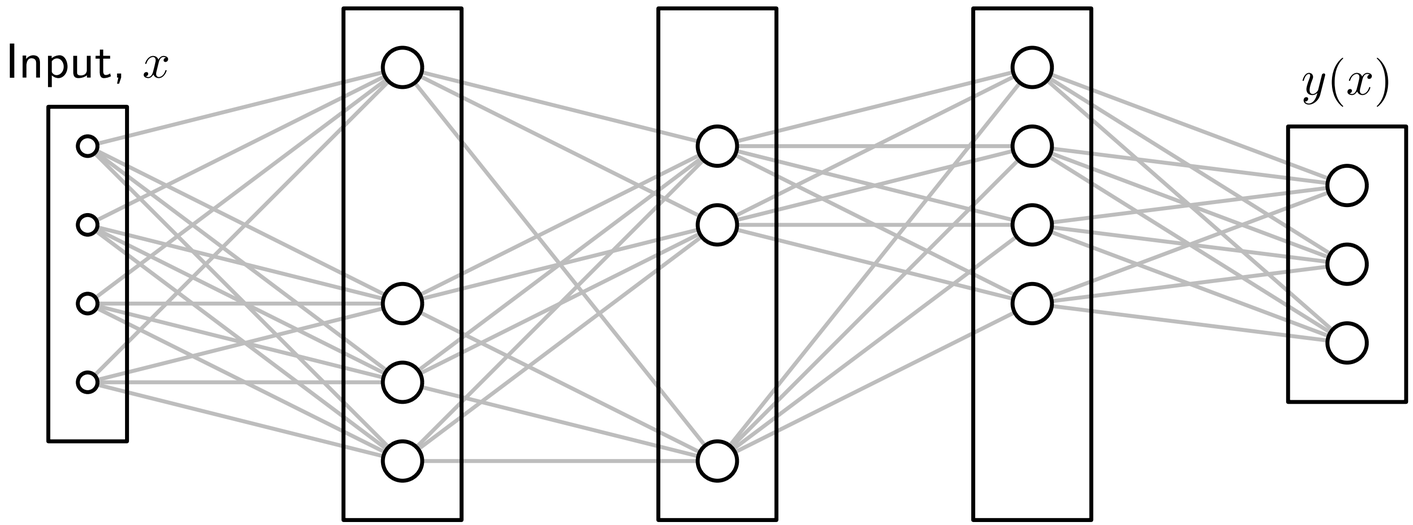
Dropout

- Example:



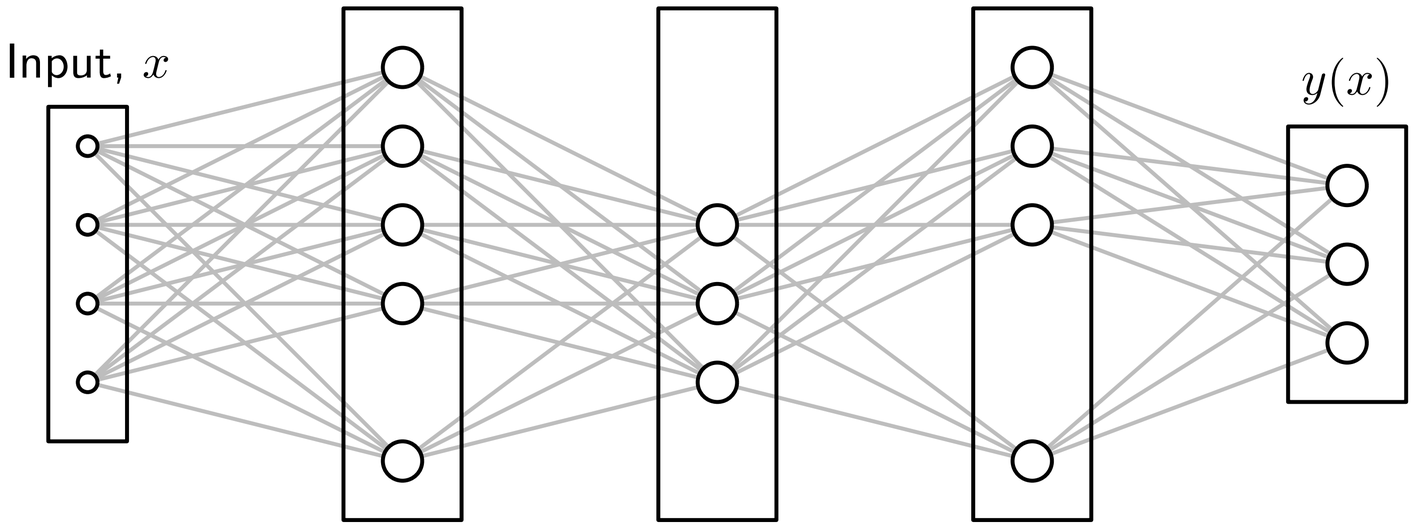
Dropout

- Example:



Dropout

- Example:



Dropout

- Dropout is widely used in present day deep learning practice
- Dropout “turns off” random units during training, to make the network more robust to unit specialization
- At test time, we combine output of “all networks”
  - Output of each unit is weighted by dropout probability *p*

Some more “tricks of the trade”

- Initialization:
  - The initialization of the neural network weights can greatly affect its learning performance
  - Common practice is to initialize biases to 0, other weights using random values
  - For ReLU units, use small random positive number
  - For sigmoid units, use “Glorot initialization”

Weights for layer ℓ selected uniformly in ($-t$, t), with

$$t = \frac{\sqrt{6}}{\sqrt{M_\ell + M_{\ell-1}}}$$

Some more “tricks of the trade”

- Hyper-parameter tuning:
  - Optimizing the performance of the network often envolves selecting several hyper-parameters (n. layers, size of each layer, dropout, regularization coefficients, etc.)
  - Hyper-parameter tuning refers to the process of fine-tuning such parameters, often by trial-and-error
    - Common approaches: grid search, random search, Bayesian optimization

Some more “tricks of the trade”

- Other tricks:
  - It’s usually a good practice to normalize the data to zero mean and unit variance
  - In stochastic gradient descent, decaying the learning rate often helps convergence
  - To help debug, you can choose a small subset of data (e.g., 50 samples) and check if your model overfits in this set (it should)
    - If not, check if units are saturated from early training
    - If loss goes up and down during training, try decreasing the learning rate

To conclude…

Introduction to neural networks and deep learning

- **Key messages:**
  - Deep learning brought about many successes in artificial intelligence
  - Works by turning learning problems into supervised learning problems, and solving these using neural networks
  - Building a deep learning systems involves (among other things) designing the network and coming up with an adequate loss function

To conclude…

Introduction to neural networks and deep learning

- **Key messages:**
  - Neural networks are trained by gradient descent, using backpropagation to compute the gradients efficiently
  - Fine-tuning a neural network to a particular problem involves a lot of trial-and-error

References

- Davis, S., and Mermelstein, P. “Comparison of parametric representations for monosyllabic word recognition in Continuously Spoken Sentences.” *IEEE Trans. Acoustics, Speech, and Signal Processing*, 28(4):357-366, 1980.
- Farfade, S., Saberian, M., and Li, L. “Multi-view face detection using deep convolutional neural networks.” In *Proc. 5th ACM Int. Conf. Multimedia Retrieval*, pp. 643-650, 2015.
- Glorot, X., and Bengio, Y. “Understanding the difficulty of training deep feedforward neural networks.” In *Proc. 13th Int. Conf. Artificial Intelligence and Statistics*, pp. 249-256, 2010.
- Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., and others. “Highly accurate protein structure prediction with AlphaFold.” *Nature*, 596:583-589, 2021.
- Lowe, D. “Object recognition from local scale-invariant features.” In *Proc. 7th IEEE Int. Conf. Computer Vision*, pp. 1150-1157, 1999.
- Minsky, M., and Papert, S. *Perceptrons*, MIT Press, 1969.

References

- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. “Language models are unsupervised multitask learners.” Technical report, OpenAI, 2019.
- Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., and Chen, M. “Hierarchical text-conditional image generation with CLIP latents.” arXiv:2204.06125, 2022.
- Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., and others. “Mastering Atari, Go, chess and shogi by planning with a learned model.” *Nature*, 588:604-609, 2020.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. “Dropout: a simple way to prevent neural networks from overfitting.” *J. Machine Learning Res.*, 15(1):1929-1958, 2014.
- Vinyals, O., Babuschkin, I., Czarnecki, W., Mathieu, M., Dudzik, A., Chung, J., and others. “Grandmaster level in StarCraft II using multi-agent reinforcement learning.” *Nature*, 575:350-354, 2019.

References

- Wu, Y., Shuster, M., Chen, Z., Le, Q., Norouzi, M., Macherey, W., and others. “Google's neural machine translation system: Bridging the gap between human and machine translation.” arXiv:1609.08144, 2016.
- Xiong, W., Droppo, J., Huang, X., Seide, F., Seltzer, M., Stolcke, A., Yu, D., and Zweig, G. “The Microsoft 2016 conversational speech recognition system.” In *Proc. 2017 IEEE Int. Conf. Acoustics, Speech and Signal Processing*, pp. 5255-5259, 2017.
- Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., and Bengio, Y. “Show, attend and tell: Neural image caption generation with visual attention.” In *Proc. 32nd Int. Conf. Machine Learning*, pp. 2048-2057, 2015.